WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Terms	Documents
L7 and generating and transforming and displaying	0

Database: US Patents Full-Text Database; US Pre-Grant Publication Full-Text Database; JPO Abstracts Database; EPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Search: L10

Refine Search | Recall Text | Clear

Search History

DATE: Thursday, May 22, 2003 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=NO; OP=OR		
L10	L7 and generating and transforming and displaying	0	L10
L9	L6 and generating and transforming and displaying	7	L9
L8	mental adj system and matrix	0	L8
L7	idea adj system and matrix	13	L7
L6	thought adj system and matrix	54	L6
L5	L1 and L2 and L3 and matrix	0	L5
L4	L1 and L2 and L3 and matrix and generating and transforming and displaying	0	L4
L3	mental adj system	5	L3
L2	idea adj system	126	L2
L1	thought adj system	145	L1

END OF SEARCH HISTORY

WEST

Generate Collection | Print

Search Results - Record(s) 1 through 7 of 7 returned.

☑ 1. Document ID: US 20020089551 A1

L9: Entry 1 of 7 File: PGPB Jul 11, 2002

PGPUB-DOCUMENT-NUMBER: 20020089551
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020089551 A1

TITLE: Method and apparatus for displaying a thought network from a thought's perspective

PUBLICATION-DATE: July 11, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Hugh, Harlan M.	Los Angeles	CA	US	
Crawford, Jenson	Los Angeles	CA	US	

US-CL-CURRENT: 345/853

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWMC | Draw Desc | Image

☑ 2. Document ID: US 20020067381 A1

L9: Entry 2 of 7 File: PGPB Jun 6, 2002

PGPUB-DOCUMENT-NUMBER: 20020067381
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020067381 A1

TITLE: Method and apparatus for organizing and processing information using a digital computer

PUBLICATION-DATE: June 6, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Hugh, Harlan M.	Los Angeles	CA	US	

US-CL-CURRENT: 345/854; 345/748

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWMC | Draw Desc | Image

☑ 3. Document ID: US 20020054167 A1

L9: Entry 3 of 7 File: PGPB May 9, 2002

PGPUB-DOCUMENT-NUMBER: 20020054167
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020054167 A1

TITLE: Method and apparatus for filtering and displaying a thought network from a thought's perspective

PUBLICATION-DATE: May 9, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Hugh, Harlan M.	Los Angeles	CA	US	

US-CL-CURRENT: 345/854

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

---

☑ 4. Document ID: US 6256032 B1

L9: Entry 4 of 7 File: USPT Jul 3, 2001

US-PAT-NO: 6256032
DOCUMENT-IDENTIFIER: US 6256032 B1

TITLE: Method and apparatus for organizing and processing information using a digital computer

DATE-ISSUED: July 3, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hugh; Harlan M.	Los Angeles	CA		

US-CL-CURRENT: 345/854; 345/764

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

---

☑ 5. Document ID: US 6166739 A

L9: Entry 5 of 7 File: USPT Dec 26, 2000

US-PAT-NO: 6166739
DOCUMENT-IDENTIFIER: US 6166739 A

TITLE: Method and apparatus for organizing and processing information using a digital computer

DATE-ISSUED: December 26, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hugh; Harlan M.	Los Angeles	CA		

US-CL-CURRENT: 345/854; 345/858, 709/100

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments KWIC | Draw Desc | Image

☑ 6. Document ID: US 6031537 A

L9: Entry 6 of 7 File: USPT Feb 29, 2000

US-PAT-NO: 6031537
DOCUMENT-IDENTIFIER: US 6031537 A

TITLE: Method and apparatus for displaying a thought network from a thought's perspective

DATE-ISSUED: February 29, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hugh; Harlan M.	Los Angeles	CA		

US-CL-CURRENT: 345/854; 345/839

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments KWIC | Draw Desc | Image

☐ 7. Document ID: EP 1103901 A2

L9: Entry 7 of 7 File: EPAB May 30, 2001

PUB-NO: EP001103901A2
DOCUMENT-IDENTIFIER: EP 1103901 A2
TITLE: Method and apparatus for analyzing thought system

PUBN-DATE: May 30, 2001

INVENTOR-INFORMATION:

NAME	COUNTRY
SUZUKI, KAZUHIKO	JP

INT-CL (IPC): G06 F 17/30

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments KWIC | Draw Desc | Image

Generate Collection | Print

Terms	Documents
L6 and generating and transforming and displaying	7

Display Format: - Change Format

Previous Page Next Page

WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Terms	Documents
L7 and generating and transforming and displaying	0

Database: US Patents Full-Text Database; US Pre-Grant Publication Full-Text Database; JPO Abstracts Database; EPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Search: L10

Refine Search

Recall Text | Clear

Search History

DATE: Thursday, May 22, 2003 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=NO; OP=OR		
L10	L7 and generating and transforming and displaying	0	L10
L9	L6 and generating and transforming and displaying	7	L9
L8	mental adj system and matrix	0	L8
L7	idea adj system and matrix	13	L7
L6	thought adj system and matrix	54	L6
L5	L1 and L2 and L3 and matrix	0	L5
L4	L1 and L2 and L3 and matrix and generating and transforming and displaying	0	L4
L3	mental adj system	5	L3
L2	idea adj system	126	L2
L1	thought adj system	145	L1

END OF SEARCH HISTORY

NPL Services for Examiners

Thursday, May 22, 2003

STIC's mission is to connect examiners to critical prior art by providing information services and access to NPL electronic resources and print collections. A STIC facility is located in each Technology Center.

Most of the electronic resources listed on these Web pages are accessed via the Internet. You must be authenticated for data to be accessed. Firewall Authentication

Specialized Information Resources for Technology Centers

Select a Technology Center

Technology Centers GO

General Information Resources

- List of Major E-Resources
- List of eBook and eJournal Titles
- General Reference Tools
- Business Methods Resources
- Bibliography of the Month
- Defensive Disclosure Resources
- Educational Methods Resources
- Legal Resources
- New Book List

General Services

- Foreign Patent Services
- PLUS System
- Request a Book or Article
- Request a Book/Journal Purchase
- Request a Prior Art Search
- Search STIC Online Catalog
- Trademark Law Library
- Translation Services

TC2100: EIC Resources and Services

Pilot of "Fast and Focused" quick turnaround searches begins on May 5, 2003! See Criteria and Search Form in 'Services' section below!

Thursday, May 22, 2003

These resources and services provide examiners with access to critical prior art. Most of the electronic resources listed on these Web pages are accessed via the Internet. You must be authenticated for data to be accessed. Firewall Authentication

■ indicates tools featured in TC's NPL training.

Information Resources

Information Resources by Class and Subclass

Databases

■ ACM Digital Library
Business Source Corporate (Corporate Resource Net)
(Multidisciplinary subject coverage)
Dialog Classic on the Web
(Training and password required.)
DTIC STINET
(Citations to Defense Technical Information Center scientific and technical documents)
EPOQUE
(EPO's databases, available on stand-alone terminal in CPK2, 4B40)
■ IEEE Xplore
(Full page images of over 800,000 Electrical & Electronic Engineering articles, papers and standards, 1988 - present. Select content is available from 1952-1987.)
INSPEC
(seven million well-indexed physics, EE, and IT abstracts, 1969-present)
Proquest Direct
(Multidisciplinary subject coverage)
Research Disclosure
(Published monthly as a paper journal and now as an on-line database product with advanced full text searching capabilities for defensive disclosure information.)
Software Patent Institute (SPI)
(Select "Free Access")
STN on the Web
(Training and password required. The other link is via the Patent Examiner's Toolkit. On your computer, click on the START button, then on the PE Toolkit, then on STN Express.)

Books and Journals

- Search STIC Online Catalog
- Books24x7
  (Computers and the Internet)
- Knovel
  (Applied science and engineering)
- NetLibrary.com
  (Multidisciplinary subject coverage)

Daily Newspapers

Fulltext newspaper articles are available electronically in Proquest Direct.

CD-ROM Resources

Older full text NPL resources/articles received in CD-Rom format. These resources are available on EIC2100 PCs in CPK2, 4B40.

Equipment

Reference Tools

- Atomica
  (Defines technical, business and other terms. Provides drawings, synonyms, related websites, and translations.)
- Bartleby.com
  (Several versions of Roget's Thesaurus, a dictionary, an encyclopedia, quotations, English usage books and more.)
- Computer References (Dictionaries, Acronyms, Encyclopedias)
- Encyclopedia Britannica
- Eric Weisstein's World of Mathematics
  (A comprehensive online encyclopedia of mathematics.)
- HowStuffWorks
  (Search a term to find articles that explain how it works.)
- Over 2000 Glossary Links
  (Links to numerous technical, specialty, and general glossaries)
- Wiley Encyclopedia of Electrical and Electronics Engineering
- Yourdictionary.com
  (Numerous "specialty dictionaries"... technological, law, business related and more.)

Services

- EIC2100 Staff
- Foreign Patent Services
- PLUS
- Request a Book/Journal Purchase
- Request a Book or Article
- Request a Prior Art Search
  [e-submit] [Printable form]
  Fast & Focused Search Criteria
- STIC Online Catalog
- Translation Services

Web Resources

- Nanotechnology
- ResearchIndex
  (Full text scientific research papers - in pdf and postscript formats.)
- Usenet Archive (Google Groups)

Submit comments and suggestions to Anne Hendrickson To report technical problems, click here

Last Modified: Friday, May 02, 2003 16:04:48

Search Results

Search Results for: [matrix<AND>((thought system))]
Found 2 of 110,178 searched. → Rerun within the Portal

Search within Results

GO > Advanced Search > Search Help/Tips

Sort by: **Title** **Publication** **Publication Date** **Score** Binder

Results 1 - 2 of 2 **short listing**

1 Hypertext and the author/reader dialogue 77%
Susan Michalak , Mary Coney
Proceedings of the fifth ACM conference on Hypertext December 1993

2 Towards multiple self-application 77%
Robert Glück
ACM SIGPLAN Notices , Proceedings of the symposium on Partial evaluation and semantics-based program manipulation May 1991
Volume 26 Issue 9

Results 1 - 2 of 2 **short listing**

Hypertext and the Author/Reader Dialogue

Susan Michalak
Department of Technical Communication
University of Washington, FH-40
Seattle, Washington 98195

Mary Coney
Department of Technical Communication
University of Washington, FH-40
Seattle, Washington 98195

ABSTRACT

Hypertext theorists tend to approach the hypertext concept from radically different philosophical positions. Some theorists stress hypertext's utility as an information storage and retrieval device; others praise hypertext's ability to free the reader from linear media; still others applaud hypertext's connectivity and its ability to create a basis for the communal creation of knowledge. The hypertext documents that these theorists envision (and create) are quite different from one another because they are based on each theorist's particular perspective on hypertext. Recently, many hypertext theorists have acknowledged a need for hypertext authors to develop a better rhetorical understanding of their readers; however, the reader roles that most hypertext theorists have thus far anticipated do not encompass all of the reader roles that hypertext can accommodate. Coney (Cone92a) has offered a comprehensive "taxonomy of readers" that—although it was originally conceived as a taxonomy of readers of conventional print—provides deeper insight into hypertext reader roles. In this paper, we will discuss the philosophical traditions invoked by various hypertext theorists and the reader roles that are accommodated or required by those traditions. Finally, we will discuss the hypertext author's virtual presence, the implied author, as a corollary of reader role.

KEYWORDS

hypertext, literary theory, implied author, rhetoric, reader roles, reader-response criticism

INTRODUCTION

Recently, hypertext theorists have acknowledged a need for hypertext authors to develop a better rhetorical understanding of their readers (Slat91, Land92a, McKn89). Slatin has pointed out that the difficulty frequently encountered by both readers and authors of hypertext documents is that "hypertext systems tend to envision three different types of readers: the reader as browser, as user, or as co-author." Slatin further states that "the relationship between these three classes can be fuzzy and therefore difficult to manage" (Slat91, 158). Part of the difficulty lies in the fact that these three categories alone do not encompass all of the reader roles that hypertext theorists have envisioned, or indeed all the roles that readers of hypertext have already adopted. Coney (1992a) has offered a comprehensive "taxonomy of readers" that—although it was originally conceived as a taxonomy of readers of conventional print—provides deeper insight into hypertext reader roles. In addition to Slatin's three categories of hypertext readers, Coney's taxonomy illuminates three other roles applicable to hypertext: *reader as receiver of information*, *reader as professional colleague*, and *reader as maker of meaning*.

The problem that Slatin cites is further compounded because different hypertext theorists and practitioners tout different aspects of the hypertext concept. Some theorists stress hypertext's utility as an information storage and retrieval device; others praise hypertext's ability to free the reader from linear media; still others applaud hypertext's *connectivity* and its ability to create a basis for the communal creation of knowledge. The hypertext documents that these theorists envision (and create) are quite different from one another because they are based on each theorist's particular perspective on hypertext. To some extent, of course, the type of hypertext document—whether fiction, poetry, manuals, etc.—influences the role a reader might play. Fiction hypertext documents, such as Michael

Joyce's *Afternoon*, provide for readers as *makers of meaning* whereas nonfiction documents, such as *1472-D* (a military standard manual that R. J. Glushko transferred into hypertext), tend to envision readers as *users* or *browsers*. However, some nonfiction hypertext documents, such as Mark Bernstein and Erin Sweeney's *The Election of 1912*, have clearly attempted to provide for readers as makers of meaning even though these documents are primarily transferring factual information. Our position in this paper is that the epistemological approach authors bring to hypertext—*regardless of their document's genre*—plays a strong role in determining the author's vision of the reader and the reader roles that result from that vision.

Much of the difficulty of determining hypertext reader roles, then, lies in the propensity of practitioners and theorists to approach hypertext from radically different—and often unacknowledged—positions. Some hypertext theorists have turned toward post-modern literary theory and others toward a social constructionist approach to knowledge to best describe the experience of hypertext. Still others adhere—perhaps unknowingly—to logical positivism in their approach to hypertext. Consequently, some clearly approach it with the perspective that meaning lies within the text, while others have the perspective that the reader creates meaning.

Our purpose in this paper is to consider how various hypertext theorists approach the author/reader relationship. We will discuss the philosophical traditions invoked by various hypertext theorists and the reader roles that are accommodated or required by those traditions. Finally, we will discuss the hypertext author's virtual presence, the implied author, as a corollary of reader role.

HYPERTEXT AND LITERARY THEORY

Hypertext theorists, such as Landow, Delany, and Bolter, have invoked literary theory, specifically deconstruction and reader-response criticism, to varying degrees in their efforts to develop a theoretical approach to hypertext. Landow has written that "What is perhaps most interesting about hypertext, though, is not that it may fulfill certain claims of structuralist and post-structuralist criticism but that it provides a rich means of testing them" (Land92a). After recognizing and naming the parallels between hypertext and literary theory, Bolter—on the other hand—does not seek to simply provide a laboratory for post-modern theory but instead calls for the development of a more positive theory of hypertext:

> Electronic writing takes us beyond the paradox of deconstruction, because it accepts as strengths the very qualities—the play of signs, intertextuality, the lack of closure—that deconstruction poses as the ultimate limitations of literature and language. . . . Deconstruction has helped to free us from a mode of thought that was too closely wedded to the technology of print. Deconstruction therefore tells what electronic writing is not. We will still need a new literary theory to achieve a positive understanding of electronic writing. (Bolt91, 166)

The Western world has frequently sought to anchor itself in some essence or concept that would provide unquestionable meaning to its language and experience. The concept employed, however, must be beyond the thought system itself; "it cannot be implicated in the very languages which it attempts to order and anchor: it must be somehow anterior to these discourses, must have existed before they did" (Eagl83, 131). Jacques Derrida, who is frequently cited by hypertext theorists, has argued that this sort of transcendental meaning is a fiction; that is, all concepts are part of an endless play of signification, "shot through with the traces and fragments of other ideas" (Eagl83, 131). Some concepts, however, are simply elevated to privileged positions based on current social ideologies. Derrida argues that if you examine such concepts closely, they can always be *deconstructed*—they are simply products of a system of meaning, rather than that which reinforces the system of meaning from the outside. Furthermore, these concepts can be defined by what they exclude. Derrida seeks to deconstruct text by focusing on what's been relegated to the periphery, or margin. Here he typically finds troublesome details that, when worked all the way through, threaten to contradict the whole of the text.

Derrida maintains that the process of de-centering is essential for intellectual change: one must dislocate the center to gain different vantage points. In print technology, a single text serves as the center of the reading experience—it provides a single vantage point and becomes the locus of reference. In contrast, hypertext as envisioned by Landow and Bolter embraces decentrality. When a text is connected with other texts, no text or idea need exist in the margin—all are equally represented and accessible. What Landow calls the metatext (the body of linked texts in a hypertext system) has "no primary axis of organization. In other words, the metatext or document set—the entity that describes what in print technology is the book, work, or single text—has no center" (Land92a, 12). Because the reader as envisioned by Landow is not locked into any pre-ordained hierarchy or organization, she makes her own interests the de facto organizing principle, or center. The text that is on-screen can be considered the center at any one time, but it is a "transient, decenterable virtual center—one created . . . only by one's act of calling up that

particular text—it never tyrannizes other aspects of the network in the way a printed text does" (Land92b, 11). As a reader traverses the metatext, she is continually shifting the center according to the needs of her investigation or experience. And, according to Landow and Delany, because no single text serves as a constant center, hypertext allows access to multiple vantage points.

By placing a text in a network of other texts, Bolter claims that hypertext embodies the "intertextuality" that Derrida envisioned: "Stressing connections rather than textual independence, the electronic space rewrites the possibilities of reference and allusion. Not only can one passage in an electronic text refer to another, but the text can bend so that any two passages touch, displaying themselves contiguously to the reader. . . . The electronic space permits us to visualize intertextuality as no previous medium has done" (Bolt91, 164-5).

By allowing the annotation of text and the creation of links between texts, "hypertext destroys one of the most basic characteristics of the printed text: its separation and univocal voice. Whenever one places a text within a network of other texts, one forces it to exist as part of a complex dialogue" (Land92b, 13). Again, hypertext mirrors literary theory by achieving the deconstructionist's vision of multivocality. The once-isolated text simply becomes one voice in many.

In their attempts to describe a model for hypertext, proponents also invoke reader-response theorists, such as Wolfgang Iser and Stanley Fish. Reader-response theory maintains that the reader actively responds to the text on a printed page; this response in turn determines the meaning of the text. Obviously, the reader is limited to some extent by the actual words on the page, but the reader is still free to exercise the imagination in interpreting the text. Rather than viewing the text as a stable, objective entity as it has traditionally been viewed, reader-response criticism maintains that the reader ultimately creates the text. The implications are that the text is dynamic: it changes from reading to reading and from reader to reader. "The objectivity of the text is an illusion, and moreover, a dangerous illusion, because it is so physically convincing" (Fish as cited in Bolt91, 157).

According to Bolter,

> The new medium [hypertext] reifies the metaphor of reader response, for the reader participates in the making of the text as a sequence of words. Even if the author has written all the words, the reader must call them up and determine the order of presentation by the choices made or the commands issued. There is no single univocal text apart from the reader; the author writes a set of potential texts, from which the reader chooses. (Bolt91, 158)

On many levels hypertext embodies much of literary theory. It deconstructs the text, provides multivocality and intertextuality, allows the readers to respond to the text (thereby creating meaning for themselves), and contextualizes the text. However, only one type of hypertext system does all of this: one that is highly extensible, allowing readers to create their own links, annotate the text, and even edit the text. Landow and Bolter are accurate in their comparison of hypertext with current literary theory, yet their underlying assumptions have created a model that is useful in assessing only one form of hypertext.

THE SOCIAL CONSTRUCTION OF KNOWLEDGE

As envisioned by Bolter and Landow and corroborated by post-modern literary theorists, hypertext denies the fixity of the text and questions the authority of the author. It allows for an aggressive, interactive reader. Hypertext reduces the authority of any one particular text by creating a type of "Rortyian conversation," where no author can serve as a "privileged expert," but all are reduced to the role of "ordinary participant." Many hypertext theorists have been trying to model hypertext on cognitive theories of knowledge to show that hypertext works the way the human mind allegedly does: through association. In his introduction to *The Society of Text*, however, Edward Barrett suggests another model for hypertext use and development, "a model that eschews cognitive terminology for an emphasis on the social construction of knowledge" (Barr89, xii). He believes that hypertext is constrained by the epistemology typically invoked by its theorists (i.e., the model of cognitive psychology). Instead, he maintains that hypertext should ideally be viewed as a "sort of topography of social construction" and goes on to describe hypertext as

> fundamentally a linguistic entity that exists to be manipulated, transformed through a series of collaborative acts either between just one user and the original database, or among many users performing various operations upon a central core of texts. . . . The hypertext focuses their interactions and initiates a new historical projection of their understanding and beliefs. . . . The programming structures underlying . . . the hypertext merely support the larger *hyper-context* of social construction: a matrix of knowledgeable peers defining what they think through the medium of language. (Barr89, xvi)

Barrett views knowledge, reality, and facts as "community generated linguistic entities" and asks us "to think of such presumably internal and individualistic possessions as thought or knowledge as essentially social in origin." The implication for hypertext is that a single individual is no longer responsible for the text—the hypertext document is socially constructed and socially owned. At issue is the rejection of the "authorial imperative." Barrett envisions a hypertext as an open-ended conversation in which authors simply offer potential texts and paths and the readers discover—and even create—what they need to know.

The notion of knowledge as social construction applied to hypertext implies a vision of hypertext as an all-inclusive medium where all voices are heard. Landow and Delany warn of the dangers of this thinking. In a model that emphasizes hypertext's inclusivity, the exclusion of a text becomes a highly meaningful act. As Landow and Delany point out, decisions about the relevance of texts and their inclusion in the system "bear heavy ideological freight" (Land92b, 31). The emphasis on inclusivity places the authors in the system on an equally accessible plane; therefore, the exclusion of a text from such a system makes that text seem much more distant than it otherwise would be.

In Barrett's social constructionist approach, the traditional relationship between author and reader is lost, and in its place we find a "community" where participants are on supposedly equal footing. Depending on the reader's/user's needs, Barrett's epistemology is almost as equally constraining as the one he eschews. If the goal of the hypertext system is to create and maintain a conversation between "knowledgeable peers," then Barrett's approach may be entirely appropriate; however, if the hypertext system is meant to provide rapid access to information to meet a reader's specific needs (that is, the reader is "needy" and thus not a peer)—as hypertext engineers such as R. J. Glushko and Robert Horn envision hypertext, then Barrett's "conversational" model of hypertext becomes irrelevant.

HYPERTEXT AND INFORMATION TRANSFER

Glushko stresses the information-transfer capability of hypertext and calls for an "engineering-like" approach to the design of hypertext documents. He coined the phrase *hypertext engineering* "to emphasize that hypertext is a means, not an end, and that using it requires a systematic and disciplined approach to task and document analysis" (Glus89). Glushko admits that his perspective diverges from the popular view of hypertext as a liberating means of presenting information but argues that his is an inevitable approach. Among hypertext theorists, Glushko is one of the most conservative, due in part to his expressed reluctance to depart from the printed medium. In his effort to apply hypertext concepts to the structure of an encyclopedia, Glushko quickly points out that the current design of encyclopedias is the result of accumulated knowledge about the organization and presentation of printed reference material. He clearly states his position:

> . . .[T]he most fundamental requirements for an electronic encyclopedia must be to exploit the structure of the printed one and to capitalize on what people know about using that structure. Only when these basic requirements have been met is there likely to be significant added value in providing new hypertext features that are possible only in electronic form. (Glus89)

Furthermore, Glushko sees benefits in having a multi-document hypertext that only has links *within* documents but not *between* them although he readily admits that his approach to multi-document hypertexts strains the definition of hypertext (Glus90). Clearly, Glushko takes a very pragmatic view toward the use of hypertext.

Like Glushko, Horn also stresses hypertext's utility as an information storage and retrieval mechanism. In his book, *Mapping Hypertext*, Horn defines hypertext as

> a form of organizing text in computers that permits the linking of any place in text (or other media) to any other place and the rapid retrieval of information by following trails of these associative links. (Horn89)

In keeping with this definition, Horn has developed a "methodology for analyzing, organizing, writing, sequencing, and formatting information to improve communication" called *Information Mapping*. His methodology provides conceptual tools for chunking, labelling, and connecting information. Horn further provides an information typology along with "'engineering-like' guidelines for the construction of information blocks" (Horn89, 85). Certainly, in his engineering-like approach, Horn is quite similar to Glushko; however, Horn appears to be much less reluctant to part from the lessons of print and specifically seeks to divorce hypertext from the constraints of printed media.

Horn maintains that his method of chunking information, which he refers to as "precision modularity," is powerful in managing large amounts of information because it prevents "intuitive chunking" (i.e., dividing information into chunks without knowing exactly why one has done so).

Both Glushko and Horn have approached hypertext from an entirely different perspective than that of Landow or Bolter. Their methods assume the stability of language. By attempting to chunk or "engineer" information, one must believe in the fixity of the text—that is, the text is the source of meaning, not the reader. The concepts of de-centering, intertextuality, and multivocality are not important to Glushko and Horn; they are simply attempting to develop a means of most efficiently presenting and communicating information. If the information is presented appropriately according to these methods, there should be no need for interpretation—the meaning is right there in the text. As opposed to current literary theorists, Glushko and Horn rely implicitly on the objectivity of language.

Horn also brings into question the desirability of Derrida's "decentrality." In Information Mapping, a central organizing principle exists by necessity; as Horn himself points out, his method is meant to efficiently present a particular subject matter. That subject matter becomes the organizing principle or center; of course, the reader is able to choose his or her own path through the information, but only within a limited realm. And the reader is certainly not permitted to edit these painstakingly chunked and labeled pieces of information.

Both Horn and Glushko's methods also preserve the authority of the author. Their methods may consist of one author or possibly a group of authors working in unison to present a single subject matter within one hypertext system. Horn and Glushko envision a goal-driven reader with specific needs; unlike Barrett's reader, Horn and Glushko's reader is not a member of a group of "knowledgeable peer[s]" seeking to "define what they think"—chances are, their reader doesn't know what to think about a particular subject and is turning to an authority to provide needed information.

HYPERTEXT READERS AND THEIR RHETORICAL ROLES

In a closed hypertext document where the nodes of information and the links between them are predetermined and locked, there may be several roles available to the reader. First, the reader may simply play the role of *receiver of information.* The reader in this role is a "passive, needy, consumer of knowledge who makes no real impact on the content or meaning of the message" (Cone92a, 59). An example of a hypertext document that conceives of its readers as receivers of information might be one that results from Horn's *Information Mapping* method. As described earlier, this method provides a set of conceptual tools for chunking, labeling, and connecting information in the hypertext environment; however, the author alone performs this work, and readers are not allowed to alter the document in any way (nor are they expected to want to).

The hypertext author using Horn's method can anticipate other reader roles as well: *reader as user* and *reader as browser.* Coney describes the *reader as user* as "goal-driven" and further comments that this reader "is assumed to be uninterested in and perhaps unwilling to evaluate the intellectual (or social) bases of the information in the text" (Cone92a, 59). Similarly, Slatin writes that the *reader as user* has a "clear—and often clearly limited—purpose. He or she enters the document in search, usually, of specific information and leaves it again after locating that information" (Slat91, 159). Slatin also comments that the path of a *user* in a hypertext document is relatively predictable provided that the author has an understanding of the task domain. Conversely, the *reader as browser,* as described by Slatin, is someone who wanders aimlessly through the material, picking up and putting down bits of information as interest dictates. Consequently, it is more difficult for the author to predict this reader's pathway through the material. The author must respond accordingly and create the necessarily links to support the *reader as browser.*

The three reader roles just described—*receiver of information, browser,* and *user*—and their accompanying hypertext documents (an example of which is one produced by Horn's *Information Mapping* method) are text- or author-centered. Both the authors of these hypertext documents and the readers playing the accompanying roles assume that the meaning resides in the text and that the reader depends on the author's authority. Of course, the most determined reader can undermine the authority of the author, but the reader is then working against the author's intentions for the text—in which case, neither is likely to benefit. (One could also argue that the reader of any document creates his own meaning; however, here we are considering the *role that the author envisions* for her reader.)

Horn approaches the communication process in a highly positivistic manner: First (as previously mentioned), by attempting to chunk information, one must believe in the fixity of the text; the text is the source of meaning, not the reader. Second, by seeking to label a chunk of information, one must assume that the chunk has only one *literal* meaning and that that literal meaning can be taken out of context and still mean the same thing for every reader. The meaning of these chunks is not open for interpretation; hence, the reader is viewed as a passive receptacle for the information provided.

In the most open hypertext documents, the author envisions a reader who will create his own links, annotate the text, or even edit the text. In other words, this hypertext document anticipates that the reader will play the role of *maker of meaning*. In this scenario, "the text and the author neither control nor even create meaning; at best they provide the occasion for readers to exercise their interpretive autonomy" (Cone92a, 61). Reader-response theory maintains that the reader actively responds to the text on a printed page; this response in turn determines the meaning of the text. Although reader-response theorists differ among themselves as to the power of the reader to dictate the meaning of a text, they do share the position that the readers and their roles are critical in understanding the complexity of what is meant by text.

Coney's *reader as maker of meaning* is also a reader of conventional print, but this role applies even more dramatically to open hypertext documents: whereas the reader of a print document actively creates meaning *mentally*, the hypertext reader does so *physically* as well. The reader can change the actual document to reflect her interpretation of the material presented. She may do this by rearranging the links between nodes, by annotating the nodes, or by changing the content of the nodes. Here the author envisions the hypertext document as an open-ended conversation in which the author simply offers potential texts and paths, and the readers discover—and even create—what they need to know.

Slatin's statements also reflect post-modern theory. Slatin maintains that the informational value of a text is not simply a function of the data it contains because none of these data can be considered information until they have been contextualized, "arranged in such a way that both the significant differences and the significant relationships among them may become apparent to the intended reader. . . . This is when information becomes knowledge" (Slat91, 156). This arrangement of data is what hypertext offers. Stanley Fish would argue that context *is* what gives an utterance its meaning (Fish89). From this viewpoint then, hypertext, by placing a text in a much larger group of related texts, is in fact placing it in context; therefore, the text becomes much more meaningful.

Some hypertext theorists, however, approach the issue of context from the opposite direction. Horn's chunking and labeling method requires that language have meaning across contexts—rather than placing text in context like Landow and Bolter, he removes it from its context (i.e., chunks it), labels it, and leaves the task of "contextualizing" to the reader.

The activities of rearranging links, annotating text, or editing text also imply that the author may invoke the reader role of *co-author*. For Slatin,

> Co-authorship may take a number of different forms—from relatively simple, brief annotations of or comments on existing material, to the creation of new links connecting material not previously linked, to the modification of existing material or the creation of new materials, or both. (Slat91, 159)

This arrangement of co-authorship further implies that the *reader as co-author* is then able to invoke reader roles for other readers, one of whom may be the original author. The distinction between authors and readers becomes blurred as each gets to play the other's role.

This blurring of roles is precisely what some designers of hypertext documents seek to achieve. In his assessment of hypertext (as previously discussed), Barrett maintains that hypertext should ideally be viewed as "a sort of topography of social construction."

Coney's *reader as professional colleague* may be the sort of reader role that Barrett envisioned when he referred to "knowledgeable peers" participating in the hypertext experience. Coney describes the reader in this role "as a member of the same intellectual community as the writer, one whose opinion and approval are sought" (Cone92a, 60). In keeping with Barrett's social constructionist model of hypertext, other hypertext proponents (Landow) have described the hypertext experience as a "Rortyian conversation," where no individual serves as "privileged expert," but all are reduced to the role of "ordinary participant." Furthermore, Coney writes that in this vision of the reader's role "the exchange of information is secondary to the establishment and maintenance of community. . . . It is as much a moral vision as a communication theory" (Cone92a, 61). Barrett's vision of hypertext is indeed a moral one.

These last three reader roles—*maker of meaning, co-author, professional colleague*—have greater implications when applied in hypertext than they do in conventional print documents. As already mentioned, the *reader as maker of meaning* in hypertext has the opportunity to *literally* make his own meaning. This bears obvious import for authorship and ownership issues. Landow and Delany might argue that we have an exaggerated sense of individual ownership of text because we have simply adapted our expectations to print technology. They write, "attitudes fostered by print technology are . . . responsible for maintaining exaggerated notions of authorial uniqueness and ownership that often convey a distorted impression of our 'original' contributions. . ." (Land92b, 16). Landow and

Delany argue that these notions wouldn't exist without the physically separated fixed text of the printed book. Because of its fixed and isolated nature, the printed book made it possible for us to acknowledge the text as "something unique and identifiable as property." Furthermore, they argue, because of the cost and labor involved in printing a book, notions of intellectual property are intensified. In the hypertext envisioned by some theorists, conventional or traditional notions of intellectual property do not apply.

IMPLICATIONS FOR HYPERTEXT AUTHORS

As theorists of conventional print point out, the author projects an image that serves "to mediate between the author and the reader . . . and the author and the text" (Cone88, 163). This image is called an *implied author* by the rhetorical scholar Wayne C. Booth who explains its function for the writer and the reader:

> As he writes, [the author] creates not simply an ideal, impersonal 'man in general' but an implied version of 'himself' that is different from the implied authors we meet in other men's works . . . . Whether we call this implied author an 'official scribe,' or adopt the term . . .the author's 'second self'—it is clear that the picture the reader gets of this presence is one of the author's most important effects. However impersonal he may try to be, his reader will inevitably construct a picture of the official scribe who writes in this manner—and of course that official scribe will never be neutral toward all values. Our reactions to his various commitments, secret or overt, will help to determine our response to the work. (Boot61, 70-71)

As in conventional text, the hypertext author must project himself differently to invoke each of the reader roles described above. Indeed, each reader role implies an authorial role. Hypertext theorists, however, have paid little attention to the notion of implied author. However, some have acknowledged the author's *virtual presence*, which can be taken as roughly equivalent to the implied author of the printed text. Landow and Delany describe virtual presence as

> a characteristic of all technology of cultural memory based on writing and symbol systems: since we all manipulate cultural codes in slightly different ways, each record of an utterance conveys a sense of the individual who makes that utterance. Hypertext differs from print technology, however in several crucial ways that amplify this notion of virtual presence. (Land92b, 14)

Landow and Delany then point out that hypertext makes the presence of individual authors more available and, therefore, mutually influential. Here they are referring to documents that consist of whole texts linked to one another (for example, a document that links literature from a certain era). But there is no mention of how the hypertext author, who is specifically creating a document for hypertext, might manipulate his virtual presence—or his implied author—to achieve a certain response from the reader.

It is apparent that the hypertext author—whether consciously or not—creates an implied author. To invoke the reader roles of *receiver of information* or *user*, the author must establish his authority on the subject matter. He may do this by imparting a tone of objectivity to his reader (by using such rhetorical devices as passive voice or absence of human agency). However, in hypertext the author has another concern; he must be careful to make all linkages that are relevant to the reader but only those linkages. If the author fails to establish a necessary link to relevant material, the reader has no way of reaching that material (unlike the conventional text reader who can flip the pages to the material she needs). On one hand, if the author does not make linkages to relevant and necessary materials, he may fail to establish himself as an authority. On the other hand, if the author creates too many links that lead to uninteresting or unnecessary materials (thinking that they might be useful to some readers), the reader will quickly learn not to trust the author's judgment. Again, unlike the conventional text reader who can just skip pages containing unnecessary or irrelevant information, hypertext readers may have to wade through unnecessary blocks of information to retrieve the information they want. A reader who attempted to play the role of *user* would be thoroughly frustrated by a document in which the designer had anticipated them to play *maker of meaning*.

If the author attempts to invoke the *reader as professional colleague*, he must carefully establish himself as a member of that discourse community just as he would if writing a conventional text. Similarly, if the author attempts to provide the role of *maker of meaning* or *co-author* for his readers, his original set of nodes and links must be interesting and challenging enough to invite comment. Although he may include the appropriate text, if the appropriate links aren't made, his hypertext will fail to invoke the intended reader roles.

What is different about hypertext from conventional text is that a single hypertext document could conceivably accommodate all reader roles; readers could be presented with a choice upon entering the document as to which role they wanted to play. However, hypertext theorists tend to envision documents that operate from opposing epistemologies: one creates documents that rely on "authorial imperative," and the other creates documents that invite "social construction" of knowledge.

What is interesting is that these entirely different epistemologies result in documents that are uniformly called "hypertext." Slatin's complaint that the relationship between reader roles becomes "fuzzy" and "difficult to manage" results from the fact that it is not immediately clear what the underlying epistemology of any given hypertext document is. Hypertext authors *and* theorists should not only acknowledge a wider range of reader roles, but also acknowledge the role of the implied author and its effectiveness in invoking these reader roles. Furthermore, because the underlying epistemologies of the various approaches to hypertext result in radically different hypertext documents, both authors and theorists must acknowledge and make explicit these epistemologies if they are to effectively communicate with their real readers.

REFERENCES

[Barr89] E. Barrett, "Introduction: Thought and Language in a Virtual Environment," In *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, Ed., Massachusetts: MIT Press, 1989.

[Bolt91] J. D. Bolter, "Writing Space: The Computer, Hypertext, and the History of Writing," Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc., 1991.

[Boot61] W. Booth, The Rhetoric of Fiction, Chicago: University of Chicago Press, 1961.

[Cone92a] M. B. Coney, "Technical Readers and Their Rhetorical Roles," *IEEE Transactions on Professional Communication*, Vol. 35, No. 2 (June 1992): pp. 58-63.

[Cone88] M. B. Coney, "The Implied Author in Technical Discourse," *J. of Advanced Composition*, Vol. 5 (1988): pp. 163-172.

[Eagl83] T. Eagleton, *Literary Theory: An Introduction*, Minneapolis, Minnesota: University of Minnesota Press, 1983.

[Fish89] S. Fish, *Doing What Comes Naturally*, Durham and London: Duke Univ. Press, 1989.

[Glus90] R. J. Glushko, "Using Off-the-shelf Software to Create a Hypertext Electronic Encyclopedia," *Technical Communication*, first quarter 1990, pp. 28-33.

[Glus89] R. J. Glushko, "Design Issues for Multi-Document Hypertexts," *Hypertext '89 Proceedings*, November 1989, pp. 51-60.

[Hedd91] C. Hedden, *Hypertext Dreams and Realities*, Masters thesis, University of Washington, Seattle, Washington, 1991.

[Horn89] R. E. Horn, *Mapping Hypertext*, Lexington, MA: The Lexington Institute, 1989.

[Jayn89] J. Jaynes, "Limited Freedom," In *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, Ed., Massachusetts: MIT Press, 1989.

[Land92a] G. P. Landow, *Hypertext: The Convergence of Contemporary Critical Theory and Technology*, Baltimore and London: The Johns Hopkins University Press, 1992.

[Land92b] G. P. Landow, and P. Delany. "Hypertext, Hypermedia and Literary Studies: the State of the Art," In *Hypermedia and Literary Studies*, George P. Landow and Paul Delany, Eds., Massachusetts: MIT Press, 1991.

[McKn89] C. McKnight, et al., "The Authoring of HyperText Documents," In *Hypertext: Theory into Practice*, Ray McAleese, Ed., Norwood, NJ: ABLEX Publishing Corporation, 1989, pp. 138-147.

[Slat91] J. Slatin, "Reading Hypertext: Order and Coherence in a New Medium," In *Hypermedia and Literary Studies*, George P. Landow and Paul Delany, Eds., Massachusetts: MIT Press, 1991.

[Zapp89] J. P. Zappen, "The Discourse Community in Scientific and Technical Communication: Institutional and Social Views," *Journal of Technical Writing and Communication*, Vol. 19, No. 1 (1989): 1-11.

Citation

Conference on Hypertext and Hypermedia >archive
Proceedings of the fifth ACM conference on Hypertext >toc
1993 , Seattle, Washington, United States

Hypertext and the author/reader dialogue

Authors
Susan Michalak
Mary Coney

Sponsors
SIGGROUP : ACM Special Interest Group on Supporting Group Work
SIGIR : ACM Special Interest Group on Information Retrieval
SIGLINK : Hypertext, Hypermedia, and Web
SIGWEB : ACM Special Interest Group on Hypertext, Hypermedia, and Web

Publisher
ACM Press New York, NY, USA

Pages: 174 - 182 Series-Proceeding-Article
Year of Publication: 1993
ISBN:0-89791-624-7

doi> http://doi.acm.org/10.1145/168750.168820 (Use this link to Bookmark this page)

Note: OCR errors may be found in this Reference List extracted from the full text article. ACM has opted to expose the complete List rather than only correct and linked references.

Barr89 E. Barrett, "Introduction: Thought and Language in a Virtual Environment," In The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information, Edward Barrett, Ed., Massachusetts: MIT Press, 1989.

Bolt91 Jay David Bolter, Writing space: the computer, hypertext, and the history of writing, Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 1991

Boot61 W. Booth, The Rhetoric of Fiction, Chicago: University of Chicago Press, 1961.

Cone92a M.B. Coney, "Technical Readers and Their Rhetorical Roles," IEEE Transactions on Professional Communication, Vol. 35, No. 2 (June 1992): pp. 58-63.

Cone88 M.B. Coney, "The Implied Author in Technical Discourse," J. of Advanced Composition, Vol. 5 (1988): pp. 163-172.

Eagl83 T. Eagleton, Literary Theory: An Introduction, Minneapolis, Minnesota: University of Minnesota Press, 1983.

Fish89 S. Fish, Doing What Comes Naturally, Durham and London: Duke Univ. Press, 1989.

Glus90 R.J. Glushko, "Using Off-the-shelf Software to Create a Hypertext Electronic Encyclopedia," Technical Communication, first quarter 1990, pp. 28-33.

Glus89 R. J. Glushko, Design issues for multi-document hypertexts, Proceedings of the second annual ACM conference on Hypertext, p.51-60, November 1989, Pittsburgh, Pennsylvania, United States

Hedd91 C. Hedden, Hypertext Dreams and Realities, Masters thesis, University of Washington, Seattle, Washington, 1991.

Horn89 Robert E. Horn, Mapping hypertext, The Lexington Institute of Hospitality Careers, Chicago, IL, 1989

Jayn89 J. T. Jaynes, Limited freedom: linear reflections on nonlinear texts, The society of text: hypertext, hypermedia, and the social construction of information, MIT Press, Cambridge, MA, 1989

Land92a George P. Landow, Hypertext: the convergence of contemporary critical theory and technology, Johns Hopkins University Press, Baltimore, MD, 1992

Land92b George P. Landow , Paul Delany, Hypertext, hypermedia and literary studies: the state of the art, Hypermedia and literary studies, MIT Press, Cambridge, MA, 1991

McKn89 C. McKnight, et al., "The Authoring of HyperText Documents," In Hypertext: Theory into Practice, Ray McAleese, Ed., Norwood, NJ: ABLEX Publishing Corporation, 1989, pp. 138-147.

Slat91 John Slatin, Reading hypertext: order and coherence in a new medium, Hypermedia and literary studies, MIT Press, Cambridge, MA, 1991

Zapp89 J.P. Zappen, "The Discourse Community in Scientific and Technical Communication: Institutional and Social Views," Journal of Technical Writing and Communication, Vol. 19, No. 1 (1989): 1-11.

↑ CITINGS *4*

Douglas S. Lange, Hypermedia potentials for analysis support tools, Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots: returning to our diverse roots, p.165-166, February 21-25, 1999, Darmstadt, Germany

Glenn J. Broadhead, Using HTML frames for institutional websites, Proceedings of the 16th annual international conference on Computer documentation, p.278-285, September 24-26, 1998, Quebec, Quebec, Canada

David M. Levy, Fixed or fluid?: document stability and new media, Proceedings of the 1994 ACM European conference on Hypermedia technology, p.24-31, September 19-23, 1994, Edinburgh, Scotland

Catherine C. Marshall, NoteCards in the age of the web: practice meets perfect, ACM Journal of Computer Documentation (JCD), v.25 n.3, August 2001

↑ INDEX TERMS

Primary Classification:
I. Computing Methodologies
↳ I.7 DOCUMENT AND TEXT PROCESSING
↳ I.7.2 Document Preparation
↳ Subjects: Hypertext/hypermedia

Additional Classification:
H. Information Systems
↳ H.1 MODELS AND PRINCIPLES
↳ H.1.2 User/Machine Systems
↳ Subjects: Human factors

J. Computer Applications
↳ J.5 ARTS AND HUMANITIES
↳ Subjects: Literature

General Terms:
Design, Human Factors, Theory

Keywords:
hypertext, implied author, literary theory, reader roles, reader-response criticism, rhetoric

↑ Peer to Peer - Readers of this Article have also read:

M^4: a metamodel for data preprocessing
Proceedings of the fourth ACM international workshop on Data warehousing and OLAP
Anca Vaduva , Jörg-Uwe Kietz , Regina Zücker

Editorial pointers
Communications of the ACM 44, 9
Diane Crawford

News track
Communications of the ACM 44, 9
Robert Fox

Forum
Communications of the ACM 44, 9
Diane Crawford

Practical programmer: of model changeovers, style, and fatware
Communications of the ACM 44, 9
Robert L. Glass

Towards Multiple Self-Application

Robert Glück
Technische Universität Wien
Institut für Computersprachen
A-1040 Vienna, Austria

Abstract

The driving force behind the investigation of program specialization and self-application are the Futamura projections which state that self-applicable program specializers, in particular, partial evaluators, may be used for transforming language specifications, in the form of interpreters, into corresponding compilers.

We propose to use a different and more general principle as the basis of self-application: Turchin's principle of metasystem transition. We demonstrate that the Futamura projections result from the application of this principle. By using the principle of metasystem transition, which allows an arbitrary number of self-applications, new products of self-application are feasible. A systematic formalization of the metasystem transition is presented and the use of a metacode as an essential element of self-application is stressed.

A concrete realization of the metasystem transition is given in Lisp. A partial evaluator without binding time analysis for a statically scoped subset of Lisp was implemented which was used successfully for self-application. Figures from initial experiments with multiple self-application are reported.

1. Introduction and Overview

Program specialization is a program transformation technique based on specializing a program with respect to some known values of its input parameters. The method of program specialization has received growing attention during the past years because of its promising applications [1]. It was independently discovered that self-applicable program specializers, in particular partial evaluators, may be used for compilation, compiler generation and compiler generator generation [7, 8, 24]. The theoretical basis for these applications is given by the three *Futamura projections* which define how these products can be obtained by self-application of a program specializer. A long time passed between the discovery of the possibilities of self-application and their realization in practice.

Several projects were initiated in the seventies and large partial evaluators were constructed. These projects gave no conclusive results regarding self-application. A non-trivial, self-applicable partial evaluator was devised by Jones, Sestoft and Søndergaard [15, 22] which was successfully used for the first time to realize all three Futamura projections in the machine.

In this system the use of a *binding time analysis* (BTA) prior to program specialization was introduced. The use of a BTA is seen as *the* key to successful self-application. This argument has been supported by the evidence that *all* self-applicable partial evaluators so far constructed are based on the use of a separate BTA [2, 4, 14, 16, 21, etc.]. One must ask which principles are sufficient to construct non-trivial, self-applicable specializers. A conclusive answer to this question is of practical as well as theoretical interest, in order to determine the principal requirements and to answer open problems of program specializers, such as whether multiple self-application can be performed at all.

In section 2 we argue that the formalization of Turchin's principle of metasystem transition provides a general and precise framework for self-application. We interpret the Futamura projections as an application of the principle of metasystem transition and examine the metasystem transition for a different number of parameters.

In section 3 a realization of the metasystem transition is given in Lisp. We outline a simple partial evaluator and give a surprisingly simple solution to the BTA requirement which explains why inefficient results were produced by self-applicable partial evaluators without BTA. We argue that the partial evaluator did perform correctly. Implementation issues of a non-trivial, self-applicable partial evaluator without BTA are discussed.

Section 4 reports results from initial experiments with multiple self-application that go beyond the Futamura projections.

2. Metasystem Transition

In this section we define Turchin's principle of metasystem transition. A familiarity with the Futamura projections is advantageous. A comprehensive presentation can be found in [16]. The relation of reflection and partial evaluation is discussed in [6].

Section 2.1 gives an introduction to the principle of metasystem transition. In section 2.2 the essential elements of metasystem transition are laid down. In section 2.3 a metasystem transition scheme for n parameters is defined. Section 2.4 concludes by examining metasystem transitions for a different number of parameters and interpreting the Futamura projections as a special case of metasystem transition.

2.1 The Principle of Metasystem Transition

A *metasystem* is a system that integrates, controls and processes other systems as objects. A *metasystem transition* (MST) is a transition from a system S to a metasystem S' which incorporates the original system as object. The principle of MST allows an unlimited number of transitions by which multi-level systems are established.

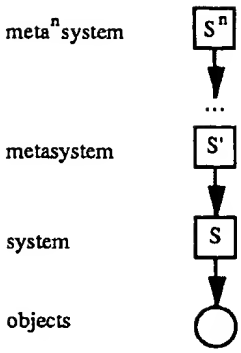

Fig. 1: The principle of metasystem transition.

In case the metasystem S' is the same as system S, that is, S' is a copy of S, the metasystem transition is referred to as *self-application* of S.

The principle of MST is seen as the essence of evolutionary processes [23] and as a powerful instrument of creative, human thinking. The formalization of MST has implications for the basis of mathematics [28] and it is expected that the realization of the MST in the machine has far reaching consequences.

> *"... to make essential progress in programming systems and artificial intelligence, one must formalize and harness the notion of metasystem transition."* [24, p.49]

2.2 The Formalization of Metasystem Transition

Now we look into the formalization of the MST in the computer. Program specialization includes a MST: a program (system) is the object of another program (metasystem). The Futamura projections define single and double self-application of a program specializer, a special case of MST. The significance of self-application of a partial evaluator was first formulated by Futamura [8] who showed that applying this principle to executable specifications of programming languages leads to remarkable results: the generation of compilers and compiler generators.

First we define the fundamentals and formulate the metacode, metavariables and metasystem. The notation of the MST is a revision of its original formulation [25, 26] and a modified form of [10]. A formalization of MST is used in the Refal supercompiler [27].

Programs describe symbolic processes which are executed by a machine. They are written in a programming language that is identified with an *abstract machine* (semantics function, execution rule, metamachine).

Let Σ, Δ, Λ be three alphabets: Σ the program alphabet, Δ the input alphabet and Λ the output alphabet. The set of all words from an alphabet (including the empty word) is designated by *. Further, let L be a programming language over Σ^*, $s \in \Sigma^*$ an L-program, $d \in \Delta^*$ the input (data, values) and $r \in \Lambda^*$ the output.

Definition: Let s be an L-program and d its input, then the output (if it exists) is denoted by the following application of the L-machine $< >_L$

$$<s\ (d)>_L$$

- In case an expression contains only applications of the same L-machine, only the outermost application is indexed by L.

$$<\ldots<\ldots<\ldots>_L\ \ldots>_L\ \ldots>_L \equiv <\ldots<\ldots<\ldots>\ldots>\ldots>_L$$

Metavariables

By doing a transition from a system S to a metasystem S' the original system S becomes the object of analysis for the metasystem S'. The original system S comprises a program p and its input d. Some parts of the analyzed system S (e.g. some of its input) may be replaced by *metavariables* which allows the analysis of the original system under generalized circumstances. These variables are named metavariables because they range over expressions of system S. Let $V_i \in \Omega^*$ be metavariables (with $\Omega^* \cap \Delta^* = \varnothing$). The following requirements have to be observed when realizing MST in the machine:

- Realization of system S and metasystem S' on the machine within the same programming environment so that, as a result, both input sets are identical.
- Unrestricted number of metavariables on one MST-level.
- Allowance for an unlimited number of transitions through the principle of MST which also requires an unlimited number of distinct metavariable sets.

Metavariables have to be distinct from any part of system S, otherwise, it would not be possible to distinguish which object is part of system S and which ranges over parts of system S. In a MST two basically different sets of words are involved:

a) The set of words in which the system S is realized, that is, the programming language L and the input $d \in \Delta^*$.
b) The set Ω^* of metavariables.

Metacode

In a MST a unique mapping is necessary which maps the original system *and* the metavariables to the input set of the metasystem. The mapping $\downarrow: L \cup \Delta^* \cup \Omega^* \rightarrow \Delta^*$ of an expression to the input set of its metasystem is called *metacoding*. The metacoding $\downarrow$ is sometimes called *downgrading*, because in the transition to a metasystem the originally autonomous system becomes an object.

A mapping has to be defined when working with specific programming languages. Without specifying a concrete metacoding, we name the result of metacoding the expression e the *metacode* of e and write $\downarrow(e)$. Such a mapping must satisfy two requirements [27]:

1. $\downarrow$ is homomorphic with respect to the concatenation of subexpressions: $\downarrow(e_1e_2) = \downarrow(e_1)\downarrow(e_2)$
2. $\downarrow$ is injective: $e_1 \neq e_2 \Rightarrow \downarrow(e_1) \neq \downarrow(e_2)$

The inverse mapping $\uparrow$ is called *demetacoding* (*upgrading*). The operation exists because the metacode is injective. Therefore: $\uparrow(\downarrow(e)) = e$.

It would be advantageous if expressions of the original system did not change at all to save the coding. Because of the required uniqueness of the mapping this is not possible [27].

Theorem: No metacoding $\downarrow$ exists, which allows an identical mapping of expressions $d \in \Delta^*$ to Δ^*.

Proof: Suppose a metacoding $\downarrow$ exists, which leaves expressions $d \in \Delta^*$ unchanged. Let $e \in \Omega^*$ be an expression of metavariables. Then the metacode $\downarrow(e) \in \Delta^*$ is an expression for which $\downarrow(e) = \downarrow(\downarrow(e))$. Therefore, the metacodes of $\downarrow(e)$ and e, for which $e \neq \downarrow(e)$, are identical. This contradicts point 2.

We conclude, that any formalism which involves a MST must formalize the notion of metacoding. If this notation is absent or ambiguous a confusion of metasystem levels or other undesirable effects may result (compare section 3.3). The representation of static and dynamic values in partial evaluation corresponds to a concrete metacode. Note that the metacoding realizes a Gödel numeration of programs.

- The following conventions will be used to avoid parenthesis when writing the metacode of an expression:

$$\downarrow(<\ldots>) \equiv \downarrow<\ldots> \quad \text{and} \quad (\downarrow(e)) \equiv \downarrow(e)$$

- Repeated metacoding is abbreviated by:

$$\uparrow^0(e) \equiv e \quad \text{and} \quad \uparrow^i(e) \equiv \uparrow_1(\uparrow_2(\ldots\uparrow_1(e))\ldots))$$

Metasystem

The metasystems we look at are program specializers, in particular, partial evaluators [9, 16] or supercompilers [27]. In the following section we will give the necessary theoretical definitions of program specializers. A discussion of related theoretical aspects may be found in [13].

A S→R-specializer α accepts an input program s (*subject program*), the known (*static*) values c1...cm and the meta-variables *D1...Dn* representing unknown (*dynamic*) values. The specializer then produces an output program r (*residual program*), which produces the same result when applied to the values d1...dn as the original program s applied to all values c1...cm and d1...dn.

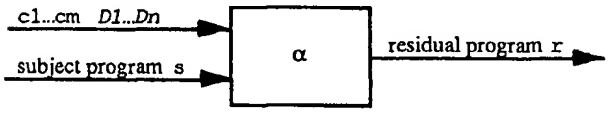

Fig. 2: Specializer α.

The subject programs accepted by an S→R-specializer are written in the *subject language* S. The residual programs are expressed in the *residual language* R. The language in which the specializer α is written is the *implementation language* I of α (figure 2).

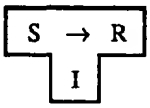

Fig. 3: T-diagram of an S→R-specializer.

In case s = α we speak of *self-application*. It is easy to verify that in the case of self-application S = I is required for α (or α is written in a subset of S). Note that the expression <s ...> is metacoded in the following definition and that s represents the text of the program.

Definition: An I-program α is an *S→R-specializer* iff for all S-programs s, values c1...cm and d1...dn and the metavariables *D1...Dn*:

$$\langle s\ (c1...cm,d1...dn)\rangle_S = \langle\langle \alpha \downarrow\langle s\ (c1...cm,\ D1...Dn)\rangle_S\rangle_I\ (d1...dn)\rangle_R$$

Consequently, the characteristic equation of specialization holds:

$$\langle s\ (c1...cm,d1...dn)\rangle_S = \langle r\ (d1...dn)\rangle_R \quad \textit{where}\ r = \langle \alpha \downarrow\langle s\ (c1...cm,\ D1...Dn)\rangle_S\rangle_I$$

Two special cases follow directly from the definition of a program specializer:

- **S→R-compilation** (m=0):

$$\langle s\ (d1...dn)\rangle_S = \langle r\ (d1...dn)\rangle_R \ \textit{where}\ r = \langle \alpha \downarrow\langle s\ (D1...Dn)\rangle_S\rangle_I$$

- **S-interpretation** (n=0):

$$\langle s\ (c1...cm)\rangle_S = \langle r\rangle_R \ \textit{where}\ r = \langle \alpha \downarrow\langle s\ (c1...cm)\rangle_S\rangle_I$$

In the first case, the S→R-compilation, the subject program is translated to the residual language R. Depending on the specialization techniques used in α, an optimization of the subject program may be feasible (e. g. by *driving* [27]). If S = R, then this may be denoted as *S-optimization*.

In the second case, the S-interpretation, the subject program is completely evaluated at specialization time and the result is expressed as constant in the language R.

Few non-trivial, self-applicable S→R-specializers have been implemented (e. g Amix [12], where S = annotated subset of Lisp and R = assembler-like language for a stack machine). For most specializers S = R where S = subset of a high-level language like Lisp or Prolog.

2.3 Multiple Metasystem Transition (appx. fig. 10)

We will present a general scheme for the formulation of *multiple metasystem transition* for a subject program with n parameters. This scheme is called the *MST-scheme for n parameters*. It is developed by repeated application of MST, by which we climb up n+1 steps of a 'MST-staircase'.

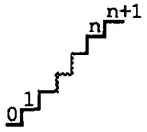

Fig. 4: MST-staircase.

Let s be an S-program with n parameters and let $v_1...v_n$ be the corresponding values. Let α be an S→R-specializer written in S, so that α is self-applicable. The MST-scheme then comprises n+1 formulas for n+1 transitions (figure 10 in the appendix).

Note: With ' we will distinguish the specializers on different metasystem levels. By definition self-application requires that the specializers α', α'' etc. be identical. However, this is not required in principle as long as the metasystems satisfy the definition of a program specializer. But it is economical to re-use the same α since for all metasystem transitions only one α must be constructed. It is also interesting to note that the outermost specializer need not be self-applicable at all.

1st Metasystem Transition

The 1st MST is given by the definition of a program specializer. An MST-formula can be interpreted as *two-dimensional* scheme by moving the expressions up and down in correspondence to ↓ and ↑. The 2-D style has the advantage that it is easy to see to which metasystem level each subexpression belongs. We say that program s belongs to level 1 whereas the metasystem α belongs to level 2 etc. The formula of the 1st MST then takes two lines.

2nd Metasystem Transition

The 2nd MST-formula follows from the 1st MST by a repeated application of a MST. The specializer α'' is applied to the definition of r_1 in the 1st MST. In this expression the metacoded value v_2 is replaced by the metavariable V_2.

Suppose the value v_2 is directly replaced by the metavariable V_2, then the expression $\downarrow\langle s\ (V_1\ V_2 v_3...v_n)$ results. The metavariable V_2 would be metacoded and belong to the same metasystem level as the variable V_1. But our intention is that the metavariable V_2 belongs to α'' and not α'. Therefore, it is necessary to lift V_2 to the level of α'', which is achieved by inverting the second metacoding of V_2 with ↑.

ith MST

The metasystem transitions can be repeated as long as some values v_i can be replaced by metavariables ($1< i \leq n$). The i^{th} formula states the relation between the $i\text{-}1^{st}$ and i^{th} level of MST-staircase. That is, the i^{th} metasystem is added.

n+1st MST

After the n^{th} MST is done all values v_i have been replaced by metavariables. In the last step the subject program s is replaced by a metavariable. By this the tower of metasystem transitions is analyzed with respect to an unknown subject program.

2.4 Applications of the Metasystem Transition Scheme

We investigate some selected applications by examining the MST-schemes for a different number of parameters. The corresponding formulas are shown in the figures 11, 12 and 13 in the appendix.

MST-scheme for 1 parameter (appx. fig. 11)

For the formulation of the MST-scheme for 1 parameter we have two possibilities of forming the 1st MST: we can either replace the value by a metavariable and assume the value is unknown at specialization time, or we let it be known at specialization time. These are the already defined cases: S→R-compilation and S-interpretation (section 2.2).

Because the subject program has 1 parameter (n = 1) the MST-scheme defines one more MST which is derived from the n+1st MST. For each of the above cases there is a 2nd MST.

- In the case of the S→R-compilation we produce by the second MST an S→R-compiler from the S→R-specializer. We call this case the *compiler extraction* from α.
- In the case of the S-interpretation we produce an S-interpreter from the S→R-specializer by the second MST. We call this case the *interpreter extraction* from α.

MST-scheme for 2 parameters (appx. fig. 13)

In the formalism of the MST-scheme the Futamura projections (FMP) are classified as MST-scheme for two parameters.

Futamura was the first to discover in 1971 that the *translation* of L-programs can be achieved by partially evaluating a language specification in the form of an L-interpreter [8]. He saw that by partial evaluation of a partial evaluator (*self-application*) the automatic generation of compilers from interpreters is possible, even the generation of a compiler generator. These products are specified by the 1st, 2nd and 3rd FMP, the driving force behind the investigation of program specialization and self-application.

The projections are reviewed in the notation of the MST. Let us examine a subject program with 2 parameters. As illustration an interpreter is used.

Definition: An S-program `int` is an *L-interpreter* iff for all L-programs `prog` and arbitrary values `data`

$$\langle \texttt{prog (data)} \rangle_L = \langle \texttt{int (prog, data)} \rangle_S$$

According to the MST-scheme we obtain the 1st and 2nd FMP from the 1st and 2nd MST by instantiating the parameter n with 2. The 3rd FMP is an instance of the n+1st MST. Therefore, between the 2nd and 3rd FMP an arbitrary number of MST formulas is 'hidden'. The relation of the FMP and the MST-scheme is illustrated in the following figure 5.

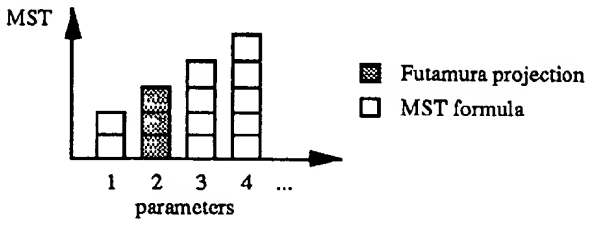

Fig. 5: Futamura Projections and MST-schemes.

MST-scheme for 3 parameters (appx. fig. 12)

Let us examine a subject program with 3 parameters. As illustration a metainterpreter is used (see definition below). If we formulate the 1st and 2nd MST in analogy to the 1st and 2nd FMP, that is, the parameter `int` of the metainterpreter is fixed in both formulas, then both transition produce results identical to the 1st and 2nd FMP. The metainterpreter plays the role of the interpreter. The products are a target program `target` and a compiler comp. Because we investigate a MST-scheme for 3 parameters a 3rd and 4th MST exist.

Definition: An S-program `mint` is an M-metainterpreter iff for all L-programs `prog`, all L-interpreters `int` and values `data`

$$\langle \texttt{prog (data)} \rangle_L = \langle \texttt{int (prog, data)} \rangle_M = \langle \texttt{mint (int, prog, data)} \rangle_S$$

3rd Metasystem Transition

The results on both sides of the equation are the same, namely an L→R-compiler `comp`. In other words: the L-interpreter `int` is converted into a compiler. The residual program produced by the application of α''' can be called a *compiler generator* $\texttt{cogen}_M$ (which accepts L-interpreters written in the language M) and the specialization a *generation of a compiler generator*.

Note that the compiler generator `cogen` produced by the 3rd FMP is fixed to the subject language S of α as generator language; that is, the specification `int` of the language L has to be written in the S. On the other hand the compiler generator $\texttt{cogen}_M$ which is produced by the 3rd MST accepts a specification of L written in M. In principle compiler generators with arbitrary generator languages M may be generated by supplying appropriate meta-interpreters.

4th Metasystem Transition

The 4th MST follows from the n+1st MST. The results on both sides of the equation are the same, namely the compiler generator $\texttt{cogen}_M$. The residual program produced by the application of α'''' can be called a *compiler-generator generator* `cogengen` and its production the *generation of a compiler-generator generator*.

3. Realization of Metasystem Transition

In this section we present a concrete realization of the MST for a specific programming system. In the field of program specialization many projects are undertaken with functional programming languages. A simple and well-known language for that purpose is Lisp, which has been chosen for the following investigations. The reader is invited to apply the principle of MST to other programming languages. For reasons of simplicity we will investigate an S→S-specializer where S is a statically scoped, first order subset of Lisp.

In section 3.1 a naive partial evaluator and its metacode are presented. In section 3.2 the metasystem transitions are formulated for a subject program with 2 parameters. In section 3.3 we investigate the requirement of binding time analysis. The solution to this problem explaining why partial evaluators returned inefficient results without BTA is given. In section 3.4 implementation issues of a partial evaluator based on this insight are discussed.

3.1 Partial Evaluation and Metacode

Notation: Subject programs are written in a first-order subset of *Scheme*, a dialect of Lisp [20].

Algorithms are written in a syntactic sugared version of McCarthy's M-expressions which can be converted to S-expressions [17]: Function names and variable names use lowercase letters. The arguments of functions are bound by square brackets and separated by semicolons. S-expressions stand for themselves and use capital letters. They are regarded as constants.

We will use the following notational device using simple pattern matching to write the algorithms concisely:

```
case ::= [case M-exp of
          pattern_1 → M-exp_1;
          ...
          pattern_n → M-exp_n;]
```

The patterns (`pattern`$_i$) describe S-expressions, e.g. `(QUOTE x)`, where variables match arbitrary S-expressions. The evaluation of the `case` is conventional: The patterns are tried sequentially, beginning with the first. Additional predicates (M-expressions) may be used in a pattern (e.g. `x, atom? [x]`). Note: Upper and lower case forms of letters are not distinguished in Scheme and therefore not distinguished in the matching with pattern constants.

A naive partial evaluator

The main function of the partial evaluator is the function `peval` to specialize an expression of the subject program (figure 6). The function takes four arguments: an expression `exp` of the subject program, a symbolic environment, represented by the lists of variable names `nam` and variable values `val`, and a list `def` containing the function definitions of the subject program. The subject expression `exp` is partially evaluated with respect to the symbolic environment represented by `nam` and `val`. The result of partially evaluating the subject expression is an expression in residual language.

Expressions of the subject program are evaluated call-by-value (applicative order). We assume function calls that should be specialized are marked as 'residual' according to some strategy (e.g. *dynamic conditionals* [2]).

Symbolic values are represented by S-expressions. For example, the expressions `(quote (A B C))` and `(cons 'A (cdr (var x)))` are symbolic values. A symbolic value of the form `(quote ...)` is called *static*, all other expressions represent *dynamic* values. The result of partially evaluating an expression is either a static or dynamic value.

The function `pe-car` for the partial evaluation of the primitive operator `car` is given below. Other functions for the partial evaluation of primitive operators (e.g. `cdr, cons`) are analogous to `pe-car`.

```
peval [exp; nam; val; def] =
  [case exp of
   x, atom? [x] → fetch-value [x; nam; val];
   (QUOTE x)    → exp;
   (CAR x)      → pe-car [peval [x; nam; val; def]];
   (CONS x y)   → pe-cons [peval [x; nam; val; def];
                           peval [y; nam; val; def]];
   ... ]

pe-car [val] =
  [case val of
   (QUOTE x)    → (QUOTE car [x]);
   (CONS x y)   → x;
   else         → (CAR val); ]
...
```

Fig. 6: Partial evaluator.

The partial evaluator permanently faces the decision either to compute a value or to generate a dynamic value. In case all arguments of a subject expression are static, the result can be computed and is static. If this is not the case `peval` tries to reduce the expression on the basis of the available arguments. The test, whether a value is static, is performed by checking if it is of the static form `(QUOTE ...)`. Note that `pe-car` performs a local optimization by reducing dynamic values of the form `(CONS x y)` to `x`.

In the discussion of the partial evaluator we have *intuitively* used a coding to represent static values different from dynamic values. If we represent static expressions, that is, expressions without any metavariable, we use the static coding; e.g. `(quote (1 2))`. When a value contains metavariables we have to use the dynamic coding. To uniquely represent metavariables we will single out the expression `(var` *varname*`)`. For simplicity we assume no other use is made of `(var` *varname*`)`.

Example: Suppose the second value of the list is unknown, then the following metacode results:

↓`(1 2)` = `(quote (1 2))`	*static*
↓`(`*N* `2)` = `(cons (var n) (quote (2))`	*dynamic*

3.2 Metasystem Transition in Scheme (appx. fig.14)

The MST-scheme for two parameters, which corresponds to the FMP, is formulated. We give their concrete formulation in Scheme (compare this formulation to the notation used in section 2). The partial evaluator presented above is a S→S-specializer, where S is a first-order subset of Scheme. The implementation language I = S.

Let `int` be an L-interpreter. We will use the following textual abbreviations: `PROG` stands for a specific L-program, `DATA` for a fixed value, `DEFINT` for the program text of `int` and `DEFPEVAL` for the program text of the partial evaluator. Further, `peval` = `peval1` = `peval2` = `peval3`. The index of metavariables indicates to which partial evaluator the variable belongs; e.g. `d1` to `peval1`. Our specific partial evaluator has four arguments. The formulation of the MST for another number of parameters is done analogously. The formulas are shown in figure 14 in the appendix. There are several different ways of writing the formulas. We have tried to minimize the effect of metacoding and to factor out parameters for later use. The reader is invited to find alternative representations of the MST-formulas.[1]

1st Futamura Projection

The formulation of the 1st MST is simple. It is the partial evaluation of `int` with respect to the fixed program `Prog` and unknown data `Data`. The variable `p1` and `d1` are put into the symbolic environment. It is not necessary to replace the static value `PROG` by the variable `p1` and to put its value into the environment but we single it to minimize the changes necessary by the next metacoding (having the 2nd MST in mind).

2nd Futamura Projection

The value `PROG` is now replaced by a metavariable. Therefore, the metacode of the expression containing the metavariable has to be changed (in bold print).

3rd Futamura Projection

The 3rd MST is constructed similarly to the 2nd MST. The value of `DEFINT` is replaced by a metavariable (for simplicity we assume that the name of the start function of the interpreter is fixed to `int`). Therefore the metacode of the expression containing the unknown values has to be changed (in bold print).

It is easy to see that by a repeated application of this method we are able to climb up MST-staircase to produce the products defined in sections 2.4 and 2.5 (as long as some values can be replaced by metavariables). It is essential that the tag `quote` or `var` remain available for the metasystems (partial evaluators) on each level. The following example illustrates how a small expression is transformed by the 1st and 2nd MST and the naive partial evaluator without the use of a BTA.

[1] Note that: `(list` $e_1 .. e_n$`)` ≡ `(cons` e_1 ... `(cons` e_n `'()))` and `(quote e)` ≡ `'e`

Example: Assume an interpreter for some programming language L is given which contains an expression of the form `(cons (car names) (car values))`, where `names` and `values` are two variables in the interpreter.

Suppose the interpreter `int` is specialized to some given L-program p, in which the variables `A`, `B` and `C` are declared, with unknown input. In this case the list of variables `names` in the L-interpreter is static, whereas the list of variable values `values` is dynamic.

The list of function definitions of the L-interpreter is not relevant in this example and we replace it by the textual abbreviation `DEFINT`. The 1st MST is written as follows:

```
(peval1 '(cons (car n1) (car v1))
        '(n1 v1)
        '((quote (A B C)) (var values))
        'DEFINT)
```

When running the `peval` on the subject expression the operation `car` on the static value `(quote (A B C))` is performed, whereas the operation `car` on the dynamic value has to be suspended. The original expression is converted to the following residual expression by partial evaluation (of course the `(var ...)` in the output may be discarded by `peval`; `var` in a residual expression stands for the identity function):

```
(cons 'A (car (var values)))
```

Suppose we want to transform the subject expression by self-application. The right side of the equation in the 2nd MST defines the production of the residual program:

```
(peval2 '(peval1 '(cons (car n1) (car v1))
                 '(n1 v1)
                 (list (list 'quote n2)
                       '(var values))
                 'DEFINT)
        '(n2)
        '((var names))
        'DEFPEVAL1)
```

The evaluation of this expression produces the following residual expression (pretty-printed with `list` instead of cons):

```
(list 'cons (list 'quote (car (var names)))
            '(car (var values))))
```

Evaluating this expression with the value `(A B C)` assigned to the variable `names` *generates* another expression. We return to the result of the 1st MST-level:

```
(cons 'A (car (var values)))
```

3.3 Metacode and Binding Time Analysis

It has been argued that the self-application of a simple partial evaluator without binding time analysis (BTA) will not produce efficient and realistic results by self-application [3, 15, 19]. BTA is seen as the solution of this problem and as a requirement for successful self-application. All self-applicable partial evaluators constructed so far have indeed used a BTA prior to the specialization [2, 4, 14-16, 21, etc.]. The diagnosis of the results was that not enough information was available at specialization time to distinguish between compile time and run time actions.

> *"If binding time analysis is not applied, the generated compilers in our experiments have turned out to be typically two orders of magnitude larger, and much less efficient."* [16, p.41]

We give a constructive and surprisingly simple solution to this requirement which demonstrates that it is actually possible to perform successful self-application without BTA. The overly general residual expression produced by a partial evaluator without BTA is given below [3]:

```
[let val1 = [case names of
              (QUOTE x)  → (QUOTE car [x]);
              (CONS x y) → x;
              else       → (CAR names);]
     val2 = [case values of
              (QUOTE x)  → (QUOTE car [x]);
              (CONS x y) → x;
              else       → (CAR values);]

 in [case (val1 val2) of
     ((QUOTE x) (QUOTE y)) → (QUOTE cons [x; y]);
     else                  → (CONS val1 val2);]]
```

Fig. 7: Overly general residual expression.

Note: the self-application in the example of section 3.2 *did* produce the correct result. We face an interesting question of 'reverse engineering':

> *Can we find the appropriate MST- formula to reproduce the same overly general residual expression?*

This can be achieved by removing the distinction between static and dynamic values in the 2nd MST-formula. Let's take the example of section 3.2. Remove the static/dynamic information in the 2nd MST and replace the values by `q1` and `q2`.

```
(peval2 '(peval1 '(cons (car n1) (car v1))
                 '(n1 v1)
                 (list q1 q2)
                 '())
        '(q1 q2)
        '((var names) (var values))
        'DEFPEVAL)
```

This achieves the same overly-general result. The cause of the problem is traced back to a very small, imprecise formulation of self-application.

Here it may look like an omission which is simply corrected by putting `quote` at the right place. But as argued in section 2 the notion of metacoding and metavariables is a crucial concept. Therefore, this 'phenomenon' is just one part of a larger scheme. This experience supports the theoretical argumentation that the MST is a necessary and more precise formalism and that the notion of metacode and metavariables is essential in self-application.

3.4 The System V-Mix

The aim was to construct a non-trivial, self-applicable partial evaluator without binding time analysis that corresponds to Mix'85 [15, 22], in which BTA was introduced for the first time. The system V-Mix is a self-applicable S→S-partial evaluator without BTA that processes a statically scoped, first-order subset of Lisp. The results with V-Mix show that self-applicable partial evaluation without BTA is neither extremely slow nor are the generated compliers as inefficient as previously thought. In addition, the partial evaluator V-Mix has been used for experiments with the 1st-5th MST. These results are reported in section 4.

The original Mix'85 is a partial evaluator that requires annotations of function calls. That is, function calls which should be specialized are marked. All other function calls are unfolded during specialization. The same approach has been taken in V-Mix.

Results: Following are some results performed by V-Mix. We show benchmarks referring to the 'classical' MP-language example, a small imperative language with `if`, `while`, a block construct and lists as data structure [18, 22].

Time	secs.
`result = <mpint (exponent, data)>`	4.78
`result = <target (data)>`	0.18
`target = <v-mix (mpint, exponent)>`	0.19
`target = <comp (exponent)>`	0.05
`comp = <v-mix (v-mix, mpint)>`	2.06

Size	no. functions	cons cells
`target`	3	202
`comp`	14	2079
`mpint`	13	570
`v-mix`	43	2095

- The *run times* were obtained with Mac Scheme 2.0 running on an Apple Macintosh IIfx (Motorola 68030 at 40 MHz). Run times do not include garbage collection. Garbage collection occurred once in the generation of the compiler and took 0.1 secs. Note that the above figures *include* the time needed for 'BTA' since V-Mix is a partial evaluator without separate BTA and performs the necessary operations during specialization.
- The *size* is given as number of cons cells needed to represent the program as list. The size numbers were obtained after post unfolding calls of functions which are called only once in the residual program [2]. Post-unfolding took 0.02 secs in the case of producing `target`.
- The program `exponent` is an MP-program to compute x^y [18]. The input are two lists x and y, the output is a list of length $|x|^{|y|}$. The `result` was computed for 5^5 (=3125).

The figures compare satisfactorily to the results reported in the literature (different hardware and software taken into account) [2, 16, 18, 22]. These numbers demonstrate that a self-applicable partial evaluator without BTA is neither extremely slow, nor are the generated compilers as inefficient as previously thought.

System In Mix'85 a separate, *global* dataflow analysis (BTA) is performed prior to specialization to determine information about static/dynamic patterns of function arguments. The subject program is then annotated *locally* with the derived information. In a partial evaluator without previous binding time analysis it is essential to maintain the static/dynamic information *during* specialization. To keep V-Mix simple and self-applicable the following methods have been used.

- polyvariant, depth-first specialization
- respecialization
- configuration analysis

In *depth-first specialization* a function is specialized at the moment a call annotation is met by calling the partial evaluator recursively for this function. This way the static/dynamic information flow from arguments to the function body can be secured in self-application.

In case the partial evaluator finds a call-annotated function the function is specialized according to the *current* static/dynamic pattern (S/D pattern) of the arguments. If different S/D patterns occur in the same subject program, variants of the function with different S/D patterns are generated. This can be compared to the approach taken in [4], in which a polyvariant BTA is described. The strategy of V-Mix is in contrast to Mix'85 which determines a *single* S/D pattern for each function during BTA. The reason for not using this strategy was to avoid the need for discarding obsolete function specializations as soon as another S/D pattern is found. From this point of view Mix'85 is monovariant with regard to S/D patterns, whereas V-Mix is polyvariant with regard to S/D patterns. In case the BTA in Mix'85 re-computes a S/D pattern for a function, V-Mix starts a *re-specialization* of the function. Experience shows that in general few re-calculations are necessary in BTA, and therefore few re-specializations occur in V-Mix.

The advantage of the method is that over-generalization of arguments is avoided and results are more compilative; the disadvantage is that the size of the residual program may grow (in the worst case exponentially since a n-ary function possibly has 2^n different S/D patterns).

During specialization a *configuration analysis* is used to determine which of arguments are used for specializing a function body and which arguments remain residual. The configuration analysis may consider either purely static or partially static arguments. The second task of the configuration analysis is a static/dynamic approximation of the function result. This analysis corresponds to a 'BTA at specialization time'.

In future it is desirable to investigate automatic strategies for folding/unfolding that can be used during specialization time. A promising method is the method of dynamic conditionals [2].

Comments

a) Specializing a simple partial evaluator without BTA is similar to the problem of some algorithmic structures being well suited for specialization, while others are not. Sometimes we are able to modify the algorithmic structure of a program in such a way that it becomes 'partial-evaluator-friendly' (e.g. as done with string matching [5]). Separating BTA from a partial evaluator was taken as a solution after it was found that this part is not pe-friendly. In addition, overly-general residual programs were found. However, overly-general programs are basically a representation problem (as argued in this paper). Termination and overly-general programs are two *independent* aspects of BTA. From this point of view performing BTA is one way to avoid overly-general residual programs. Performing BTA can be regarded as one possible method of taming termination of self-application by providing clever finiteness annotations by computing generalizations beforehand.

b) The importance of a precise formulation of the MST in the machine and the necessity of carefully representing static/dynamic information in the formulation of the MST was originally found by the author in a different setting during his work on a self-applicable supercompiler [10, 11]. Analogous problems were found when self-applying a supercompiler.

4. Experiments with Multiple Metasystem Transition

In this section results of two initial experiments with multiple MST are reported, in particular, the 1st - 5th MST. To the best of the author's knowledge these are the first results with multiple self-application that go beyond the Futamura projections.

Experiments: The system used in the following simple experiments is V-Mix. The aim of these initial experiments was to demonstrate that multiple self-application can be realized in the machine. In both experiments the arguments of the subject function were moved to different MST-levels and replaced by meta-variables; e.g. in the 3rd MST three arguments were replaced by metavariables (compare 2-D representation of 3rd MST in fig. 12).

- In the first experiment a function `tuple` was used, which builds a list of 5 elements from 5 arguments. The resulting residual function body is an expression analogous to the residual expression obtained in the example of section 3.2.

- In the second experiment a recursive function `transpose` to transpose a 5 × n matrix was used. The function takes 5 parameters. Each parameter represents a line of the original matrix as a list of elements. The result of `transpose` is a list containing the lines of the transposed matrix. By self-application the algorithm is transformed into a program which takes the first line of the matrix and then produces a second program that takes the second line ... and finally returns the transposed matrix as result.

Results: The results of both experiments are shown in the figures 8 and 9. Figure 8 reports the time taken to produce the residual programs. Figure 9 shows the size of the produced residual programs. Note that the y-axis has a logarithmic scale.

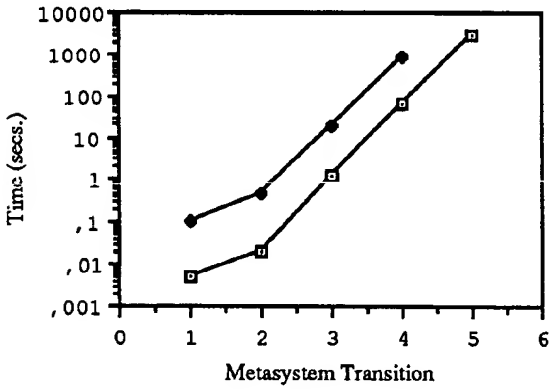

Fig. 8: Run times of multiple self-application.

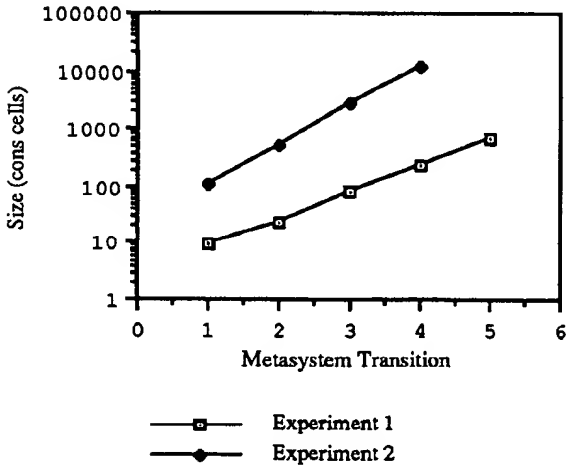

Fig. 9: Size of residual programs.

The results are not surprising. Run times increase exponentially as new levels of metasystems are added since the i^{th} metasystem is completely run by the $i+1^{st}$ metasystem. The size of the residual programs grows exponentially as new expressions to generate programs of the next lower level are added (in the program produced by the $i+1^{st}$ MST additional expressions are necessary to generate programs of the i^{th} MST-level).

Run times could be improved further by taking the following steps.

a) The evaluation of static expressions could be supported in each metasystem by `eval`-like primitive functions. That is, the computation of a static expression is passed up to the next metalevel which in turn may pass the expression up until the expression finally reaches the metamachine (e.g. Lisp interpreter). In our experiments the computation of static expressions performed on the i^{th} level is completely interpreted by the $i+1^{st}$ level.

b) Another method, that could be used to accelerate the specialization of programs is called *freezer* [29]. The freezer is a mechanism that allows the evaluation of dynamic expressions by the metamachine (e.g. Lisp interpreter). As long as an evaluation is independent of dynamic values, the machine can reduce dynamic expressions in the same way as static expressions. The evaluation of a dynamic expression is stopped as soon as a dynamic value takes part in a computation. The machine stops and returns the remaining, irreducible expression. The specializer then decides how to process this expression.

Note that in the process of passing an expression up and down in the MST-staircase metacoding and demetacoding occurs. This again stresses the argument that a precise notion of metacoding is essential in self-application. For the implementation of the freezer it is necessary to extend the machine with the notation of metavariable and metacode. The freezer was originally developed for use in the supercompiler and implemented in the Refal system [29]. In experiments it was shown that this device speeds up specialization approximately by a factor 3. Similar results can be expected in a Lisp system.

c) An alternative method to multiple self-application is *incremental self-application* (*incremental MST*). This is a method in which the MST-staircase is taken step by step by successive application of MST. The method of incremental self-application is based on the observation that a program γ (=`cogen`) resulting from the 3^{rd} FMP ($n+1^{st}$ MST with 2 parameters) can be used to convert arbitrary subject programs with two parameters into residual programs that takes the first value and produce a program that take the second value to produce the final result.

The idea of incremental self-application is to achieve the effect of multiple self-application by repeated application of γ to a subject program $\langle s\ (v_1..v_n)\rangle_s$ and by separatinghe parameters in each successive step into two groups. The program r_{i+1} is computed from r_i by separating the parameters of r_i into two groups of metavariables $V_{i+1}..V_n$ and V_i, where the first group represents the first value and V_i the second value. The following series of programs r_i ($1 \le i \le n$) is constructed by incremental self-application.

$$
\begin{aligned}
\langle s\ (v_1..v_n)\rangle_s &= \langle r_1\ (v_1..v_n)\rangle_R \\
&= \langle\langle r_2\ (v_2..v_n)\rangle\ (v_1)\rangle_R \\
&= \langle\langle\langle r_3\ (v_3..v_n)\rangle\ (v_2)\rangle\ (v_1)\rangle_R \\
&\quad \ldots \\
&= \langle\langle\langle \ldots \langle\langle r_n\ (v_n)\rangle \ldots\rangle\ (v_2)\rangle\ (v_1)\rangle_R
\end{aligned}
$$

Final remark regarding the experiments with multiple MST: The author's experience is, that as soon as a configuration like $\langle\alpha'''\ \downarrow\langle\alpha''\ \downarrow\langle\alpha'\ \downarrow\langle s\ \ldots\rangle\rangle\rangle\rangle$ has been run successfully the following transitions pose no technical difficulties (except space and run time). The reason is that in this configuration of self-application all problematic constellations of metasystems occur. Especially running the specializer α'' turned out to be subtle because α'' is analyzed by the specializer α''' and at the same time α'' analyzes another specializer α' working on a subject program `s`.

5. Conclusion

We proposed to use the principle of metasystem transition as basis of self-application by giving a formalization of the metasystem transitions and their realization on the machine. We have shown that the Futamura projections can be seen as a special case of the metasystem transition scheme for two parameters. The advantage of using the metasystem transition scheme is that new products of self-application can be defined; e.g. a compiler generator with an arbitrary generator language. We have argued that metacoding and metavariables play an essential role in self-application. By investigating the binding time requirement for self-applicable program specializers with the metasystem transition formalism, we have traced back the problem to an imprecise formulation of the self-application. We have demonstrated by using a depth-first specialization strategy and configuration analysis that a self-applicable partial evaluator without binding time analysis can be realized. In addition initial experiments with multiple self-application have been performed with the system.

This work is seen as step towards multiple and potentially unlimited self-application (restrictions of run time and space not taken into account). In particular it will be interesting to investigate the products projectable by the notation of multiple metasystem transition. However, it can be expected that partial evaluation alone may not be strong enough to satisfactorily generate all products possible by multiple self-application. No doubt it will be necessary to combine and extend partial evaluation and to investigate new and stronger methods.

Acknowledgments

The author would like to express his hearty thanks to Valentin Turchin, who led him to self-application of program specializers and metasystem transition, for his hospitality and the possibility of taking part in the supercompiler project.

The work on partial evaluation and binding time analysis was inspired by a visit to DIKU. The author would like to thank Neil D. Jones for his support and Anders Bondorf, Torben Mogensen and Peter Sestoft for their useful discussions and comments. Special thanks go to Olivier Danvy for thorough comments, to Sergei Romanenko, to Manfred Brockhaus and the anonymous referees who have contributed in many ways. Thanks also to Renate Mielacher.

References

[1] Bjørner D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. North-Holland: Amsterdam 1988.

[2] Bondorf A., Danvy O., Automatic autoprojection of recursive equations with global variables and abstract data types. Institute of Datalogy, University of Copenhagen. DIKU Report No. 90/4, 1990.

[3] Bondorf A., Jones N. D., Mogensen T. Æ. et al., Binding time analysis and the taming of self-application. Institute of Datalogy, University of Copenhagen. Report 1988.

[4] Consel C., New insights into partial evaluation: the Schism experiment. In: Ganzinger H. (ed.), ESOP '88. (Nancy, France). Lecture Notes in Computer Science, Vol. 300, 236-246, Springer-Verlag 1988.

[5] Consel C., Danvy O., Partial evaluation of pattern matching in strings. In: Information Processing Letters, 30: 79-86, 1989.

[6] Danvy O., Across the bridge between reflection and partial evaluation. In: Bjørner D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. (Gammel Avernæs, Denmark). 83-116, North-Holland 1988.

[7] Ershov A. P., On the essence of compilation. In: Neuhold E. J. (ed.), Formal Description of Programming Concepts. 391-420, North-Holland Publishing Co. 1978.

[8] Futamura Y., Partial evaluation of computation process - an approach to a compiler-compiler. In: Systems, Computers, Controls, 2(5): 45-50, 1971.

[9] Futamura Y., Nogi K., Generalized partial evaluation. In: Bjørner D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. (Gammel Avernæs, Denmark). 133-151, Elsevier Science Publishers B. V. 1988.

[10] Glück R., Turchin V. F., Experiments with a self-applicable supercompiler. City University New York. Technical Report 1989.

[11] Glück R., Turchin V. F., Application of metasystem transition to function inversion and transformation. In: Proceedings of the ISSAC '90. (Tokyo, Japan). 286-287, ACM Press 1990.

[12] Holst N. C. K., Language triplets: the Amix approach. In: Bjørner D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. (Gammel Avernæs, Denmark). 167-185, North-Holland 1988.

[13] Jones N. D., Partial evaluation, self-application and types. In: Paterson M. S. (ed.), Automata, Languages and Programming. (Warwick University, England). Lecture Notes in Computer Science, Vol. 443, 639-659, Springer-Verlag 1990.

[14] Jones N. D., Gomard C. K., Bondorf A. et al., A self-applicable partial evaluator for the lambda calculus. In: IEEE Computer Society 1990 - International Conference on Computer Languages. (New Orleans, Louisiana). 1-10, IEEE Computer Society 1990.

[15] Jones N. D., Sestoft P., Søndergaard H., An experiment in partial evaluation: the generation of a compiler generator. In: Jouannaud J.-P. (ed.), Rewriting Techniques and Applications. (Dijon, France). Lecture Notes in Computer Science, Vol. 202, 124-140, Springer-Verlag 1985.

[16] Jones N. D., Sestoft P., Søndergaard H., Mix: a self-applicable partial evaluator for experiments in compiler generation. In: Lisp and Symbolic Computation, 2(1): 9-50, 1989.

[17] McCarthy J., Abrahams P. W., Edwards D. J. et al., Lisp 1.5 Programmer's Manual. MIT Press: Cambridge, Massachusetts 1962.

[18] Mogensen T. Æ., Partially static structures in a self-applicable partial evaluator. In: Bjørner D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. (Gammel Avernæs, Denmark). 325-347, North-Holland 1988.

[19] Mogensen T. Æ., Binding time aspects of partial evaluation. University of Copenhagen, Ph.D. Thesis, 1989.

[20] Rees J., Clinger W., Revised^3 report on the algorithmic language Scheme. In: SIGPLAN Notices, 21(12): 37-79, 1986.

[21] Romanenko S. A., A compiler generator produced by a self-applicable specializer can have a surprisingly natural and understandable structure. In: Bjørner D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. (Gammel Avernæs, Denmark). 445-463, North-Holland 1988.

[22] Sestoft P., The structure of a self-applicable partial evaluator. In: Ganzinger H., Jones N. D. (ed.), Programs as Data Objects. (Copenhagen, Denmark). Lecture Notes in Computer Science, Vol. 217, 236-256, Springer-Verlag 1986.

[23] Turchin V. F., The Phenomenon of Science. Columbia University Press: New York 1977.

[24] Turchin V. F., A supercompiler system based on the language Refal. In: SIGPLAN Notices, 14(2): 46-54, 1979.

[25] Turchin V. F., The language Refal - the theory of compilation and metasystem analysis. Courant Institute of Mathematical Sciences. Courant Computer Science Report No. 20, 1980.

[26] Turchin V. F., The use of metasystem transition in theorem proving and program optimization. In: de Bakker J. W., van Leeuwen J. (ed.), Automata, Languages and Programming. (Noordwijkerhout, Netherlands). Lecture Notes in Computer Science, Vol. 85, 645-657, Springer-Verlag 1980.

[27] Turchin V. F., The concept of a supercompiler. In: ACM TOPLAS, 8(3): 292-325, 1986.

[28] Turchin V. F., A constructive interpretation of the full set theory. In: The Journal of Symbolic Logic, 52(1): 172-201, 1987.

[29] Turchin V. F., Refal-5, Programming Guide and Reference Manual. New England Publishing Co.: Holyoke, Massachusetts 1989.

Appendix

1st Metasystem Transition

$$\langle s\ (v_1 \ldots v_n)\rangle_S =$$
$$\langle\langle \alpha \downarrow \langle s\ (V_1 v_2 \ldots v_n)\rangle\rangle_S\ (v_1)\rangle_R$$

The two-dimensional interpretation is:

$$\langle s\ (v_1 \ldots v_n)\rangle_S = \langle\langle \alpha \ldots\ldots\ldots\ldots \rangle_S\ (v_1)\rangle_R$$
$$\langle s\ (V_1\ v_2 \ldots v_n)\rangle$$

Consequently, there are two ways of computing result:

result = $\langle s\ (v_1 \ldots v_n)\rangle_S$

result = $\langle r_1\ (v_1)\rangle_R$ *where* $r_1 = \langle \alpha \downarrow \langle s\ (V_1 v_2 \ldots v_n)\rangle\rangle_S$

2nd Metasystem Transition

$$\langle \alpha' \downarrow \langle s\ (V_1 v_2 \ldots v_n)\rangle\rangle_S =$$
$$\langle\langle \alpha'' \downarrow \langle \alpha' \downarrow \langle s\ (V_1 \uparrow V_2 v_3 \ldots v_n)\rangle\rangle\rangle_S\ (v_2)\rangle_R$$

The two-dimensional interpretation is:

$$\langle \alpha' \ldots\ldots\ldots\ldots \rangle_S = \langle\langle \alpha'' \ldots\ldots\ldots\ldots\ldots\ldots \rangle_S\ (v_2)\rangle_R$$
$$\langle s\ (V_1\ v_2 \ldots v_n)\rangle \qquad \langle \alpha' \ldots\ldots V_2 \ldots\ldots\ldots \rangle$$
$$\langle s\ (V_1 \ldots v_3 \ldots v_n)\rangle$$

Consequently, there are two ways of computing r_1:

$r_1 = \langle \alpha' \downarrow \langle s\ ((V_1 v_2 \ldots v_n))\rangle\rangle_S$

$r_1 = \langle r_2\ (v_2)\rangle_R$ *where* $r_2 = \langle\langle \alpha'' \downarrow \langle \alpha' \downarrow \langle s\ (V_1 \uparrow V_2 v_3 \ldots v_n)\rangle\rangle\rangle\rangle_S$

i th Metasystem Transition

$$\langle \alpha^{i-1} \downarrow \langle \alpha^{i-2} \ldots \downarrow \langle s\ (\uparrow^0 V_1 \ldots \uparrow^{i-2} V_{i-1}\ v_i \ldots v_n)\rangle \ldots\rangle\rangle_S$$
$$= \langle\langle \alpha^i \downarrow \langle \alpha^{i-1} \downarrow \langle \alpha^{i-2} \ldots \downarrow \langle s\ (\uparrow^0 V_1 \ldots \uparrow^{i-2} V_{i-1}\ \uparrow^{i-1} V_i\ v_{i+1} \ldots v_n)\rangle \ldots\rangle\rangle\rangle_S\ (v_i)\rangle_R$$

The two-dimensional interpretation is:

$$\langle \alpha^{i-1} \ldots\ldots\ldots\ldots\ldots\ldots \rangle_S = \langle\langle \alpha^i \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \rangle_S\ (v_i)\rangle_R$$
$$\langle s\ (\uparrow^0 V_1 \ldots \uparrow^{i-2} V_{i-1}\ v_i \ldots v_n)\rangle \qquad \langle \alpha^{i-1} \ldots\ldots\ldots\ldots V_i \ldots\ldots \rangle$$
$$\langle s\ (\uparrow^0 V_1 \ldots \uparrow^{i-2} V_{i-1}\ \ v_{i+1} \ldots v_n)\rangle$$

Consequently, there are two ways of computing r_{i-1}:

$r_{i-1} = \langle \alpha^{i-1} \downarrow \langle \alpha^{i-2} \ldots \downarrow \langle s\ (\uparrow^0 V_1 \ldots \uparrow^{i-2} V_{i-1}\ v_i \ldots v_n)\rangle \ldots\rangle\rangle_S$

$r_{i-1} = \langle r_i\ (v_i)\rangle_R$ *where* $r_i = \langle \alpha^i \downarrow \langle \alpha^{i-1} \ldots \downarrow \langle s\ (\uparrow^0 V_1 \ldots \uparrow^{i-2} V_{i-1}\ \uparrow^{i-1} V_i\ v_{i+1} \ldots v_n)\rangle \ldots\rangle\rangle_S$

n+1st Metasystem Transition

$$\langle \alpha^n \downarrow \langle \alpha^{n-1} \ldots \downarrow \langle s\ (\uparrow^0 V_1 \ldots \uparrow^{n-1} V_n)\rangle \ldots\rangle\rangle_S$$
$$= \langle\langle \alpha^{n+1} \downarrow \langle \alpha^n \downarrow \langle \alpha^{n-1} \ldots \downarrow \langle \uparrow^n S\ (\uparrow^0 V_1 \ldots \uparrow^{n-1} V_n)\rangle \ldots\rangle\rangle\rangle_S\ (s)\rangle_R$$

The two-dimensional interpretation is:

$$\langle\langle \alpha^n \ldots\ldots\ldots\ldots\ldots\ldots \rangle_S\ (v_n)\rangle = \langle\langle \alpha^{n+1} \ldots\ldots\ldots\ldots\ldots\ldots \rangle_S\ (s)\rangle$$
$$\langle s\ (\uparrow^0 V_1 \ldots \uparrow^{n-1} V_n)\rangle \qquad \langle \alpha^n \ldots S \ldots\ldots\ldots\ldots\ldots \rangle$$
$$\langle \ldots (\uparrow^0 V_1 \ldots \uparrow^{n-1} V_n)\rangle$$

Consequently, there are two ways of computing r_{i-1}:

$r_n = \langle \alpha^n \downarrow \langle \alpha^{n-1} \ldots \downarrow \langle s\ (\uparrow^0 V_1 \ldots \uparrow^{n-1} V_n)\rangle \ldots\rangle\rangle_S$

$r_n = \langle r_{n+1}\ (s)\rangle_R$ *where* $r_{n+1} = \langle \alpha^{n+1} \downarrow \langle \alpha^n \ldots \downarrow \langle \uparrow^n S\ (\uparrow^0 V_1 \ldots \uparrow^{n-1} V_n)\rangle \ldots\rangle\rangle_S$

Fig. 10: MST-scheme for n parameters.

1st Metasystem Transition

a) S→R-compilation (m=0): $\langle s\ (d1..dn)\rangle_S = \langle r\ (d1..dn)\rangle_R$ *where* $r = \langle \alpha \downarrow\langle s\ (D1..Dn)\rangle_S\rangle_I$

b) S-interpretation (n=0): $\langle s\ (c1..cm)\rangle_S = \langle r\rangle_R$ *where* $r = \langle \alpha \downarrow\langle s\ (c1..cm)\rangle_S\rangle_I$

2nd Metasystem Transition

a) In the case of the S→R-compilation an S→R-compiler is produced from the S→R-specializer by the 2nd MST. This case is called *compiler extraction* from α.

$\langle \alpha\ .....\rangle_S = \langle\langle \alpha\ .....\rangle_S\ (s)\rangle_R$

<s (D)> <α ..S.....>

<...(D)>

Consequently, there are two ways of computing target:

$target = \langle \alpha \downarrow\langle s\ (D)\rangle\rangle_S$

$target = \langle comp_\alpha\ (s)\rangle_R$ *where* $comp_\alpha = \langle \alpha\downarrow\langle \alpha \downarrow\langle\uparrow S\ (D)\rangle\rangle_S$

b) In the case of the S-interpretation an S-interpreter is produced from the S→R-specializer by the 2nd MST. This case is called the *interpreter extraction* from α.

$\langle \alpha\ .....\rangle_S = \langle\langle \alpha\ .....\rangle_S\ (s,\ d)\rangle_R$

<s (d)> <α .S..D..>

<..(..)>

Consequently, there are two ways of computing $result_r$. Note: $result = \langle result_r\rangle_R$

$result_r = \langle \alpha \downarrow\langle s\ (d)\rangle\rangle_S$

$result_r = \langle int_\alpha\ (s,\ d)\rangle_R$ *where* $int_\alpha = \langle \alpha\downarrow\langle \alpha \downarrow\langle\uparrow S\ (\uparrow D)\rangle\rangle_S$

Fig. 11: MST-scheme for 1 parameter.

1st and 2nd Metasystem Transition

If we formulate the 1st and 2nd MST in analogy to the FMP, that is, the parameter int of the metainterpreter mint is fixed in both formulas, then the 1st and 2nd MST produce results identical to the 1st and 2nd FMP. The metainterpreter plays the role of a plain interpreter. The products are a target program target and a compiler comp. Note that the L-interpreter int is written in M.

3rd Metasystem Transition

$\langle \alpha'' \downarrow\langle \alpha' \downarrow\langle mint\ (int,\ \uparrow Prog,\ Data)\rangle\rangle\rangle_S$

$= \langle\langle \alpha'' \downarrow\langle \alpha'' \downarrow\langle \alpha' \downarrow\langle mint\ (\uparrow\uparrow Int,\ \uparrow Prog,\ Data)\rangle\rangle\rangle_S\ (int)\rangle_R$

The 3rd MST follows from the i+1st MST. The two-dimensional representation is:

$\langle \alpha'' .....\rangle_S = \langle\langle \alpha''' .....\rangle_S\ (int)\rangle_R$

<α' .....Prog.....> <α'' .....Int.....>

<mint (int.....Data)> <α' .....Prog.....>

<mint (.....Data)>

Consequently, there are two ways of producing the compiler comp. Note that $cogen_M$ is a compiler generator with the generator language M and that the L-interpreter int is written in M.

$comp = \langle d' \downarrow\langle \alpha' \downarrow\langle mint\ (int,\ \uparrow Prog,\ Data)\rangle\rangle\rangle_S$

$comp = \langle cogen_M\ (int)\rangle_R$ *where* $cogen_M = \langle \alpha''' \downarrow\langle \alpha'' \downarrow\langle \alpha' \downarrow\langle mint\ (\uparrow\uparrow Int,\ \uparrow Prog,\ Data)\rangle\rangle\rangle_S$

4th Metasystem Transition

$\langle \alpha''' \downarrow\langle \alpha'' \downarrow\langle \alpha' \downarrow\langle mint\ (\uparrow\uparrow Int,\ \uparrow Prog,\ Data)\rangle\rangle\rangle_S$

$= \langle\langle \alpha'''\langle \alpha''' \downarrow\langle \alpha'' \downarrow\langle \alpha' \downarrow\langle\uparrow\uparrow\uparrow Mint\ (\uparrow\uparrow Int,\ \uparrow Prog,\ Data)\rangle\rangle\rangle_S\ (mint)\rangle_R$

The 4th MST follows from the n+1st MST. The two-dimensional representation is:

$\langle \alpha''' .....\rangle_S = \langle\langle \alpha'''' .....\rangle_S\ (mint)\rangle_R$

<α'' .....Int.....> <α''' .....Mint.....>

<α' .....Prog.....> <α'' .....Int.....>

<mint (.....Data)> <α' .....Prog.....>

<...(.....Data)>

Consequently, there are two ways of producing the compiler $cogen_M$:

$cogen_M = \langle \alpha'' \downarrow\langle \alpha'' \downarrow\langle \alpha' \downarrow\langle mint\ (\uparrow\uparrow Int,\ \uparrow Prog,\ Data)\rangle\rangle\rangle_S$

$cogen_M = \langle cogengen\ (mint)\rangle_R$

where $cogengen = \langle \alpha''''\langle \alpha''' \downarrow\langle \alpha'' \downarrow\langle \alpha' \downarrow\langle\uparrow\uparrow\uparrow Mint\ (\uparrow\uparrow Int,\ \uparrow Prog,\ Data)\rangle\rangle\rangle_S$

Fig. 12: MST-scheme for 3 parameters and an M-metainterpreter mint.

1st Futamura Projection

$$\langle \texttt{int (prog, data)} \rangle_S = \langle\langle \alpha \downarrow \langle \texttt{int (prog, } \mathit{Data}\texttt{)} \rangle \rangle_S \texttt{ (data)} \rangle_R$$

Consequently, there are two ways of computing `result`:

`result` = $\langle$`prog (data)`$\rangle_L$ = $\langle$`int (prog, data)`$\rangle_S$

`result` = $\langle$`target (data)`$\rangle_R$ *where* `target` = $\langle \alpha \downarrow (\langle$`int (prog, `*Data*`)`$\rangle)\rangle_S$

2nd Futamura Projection

$$\langle \alpha \downarrow \langle \texttt{int (prog, } \mathit{Data}\texttt{)} \rangle \rangle_S = \langle\langle \alpha \downarrow \langle \alpha \downarrow \langle \texttt{int } (\uparrow \mathit{Prog}, \mathit{Data}) \rangle\rangle\rangle_S \texttt{ (prog)} \rangle_R$$

Consequently, there are two ways of computing `target`:

`target` = $\langle \alpha$ `(int, prog)`$\rangle_S$

`target` = $\langle$`comp (prog)`$\rangle_R$ *where* `comp` = $\langle \alpha \downarrow \langle \alpha \downarrow \langle$`int` $(\uparrow \mathit{Prog}, \mathit{Data})\rangle\rangle\rangle_S$

3rd Futamura Projection

$$\langle \alpha \downarrow \langle \alpha \downarrow \langle \texttt{int } (\uparrow \mathit{Prog}, \mathit{Data}) \rangle\rangle\rangle_S = \langle\langle \alpha \downarrow \langle \alpha \downarrow \langle \alpha \downarrow \langle \uparrow\uparrow \mathit{Int } (\uparrow \mathit{Prog}, \mathit{Data}) \rangle\rangle\rangle\rangle_S \texttt{ (int)} \rangle_R$$

Consequently, there are two ways of computing `comp`:

`comp` = $\langle \alpha \downarrow \langle \alpha \downarrow \langle$`int` $(\uparrow \mathit{Prog}, \mathit{Data})\rangle\rangle\rangle_S$

`comp` = $\langle$`cogen (int)`$\rangle_R$ *where* `cogen` = $\langle \alpha \downarrow \langle \alpha \downarrow \langle \alpha \downarrow \langle \uparrow\uparrow \mathit{Int } (\uparrow \mathit{Prog}, \mathit{Data})\rangle\rangle\rangle\rangle_S$

Fig. 13: MST-scheme for 2 parameters and an L-interpreter `int`.

1st Futamura Projection

```
(int 'PROG 'DATA) = ((peval1 '(int p1 d1)
                             '(p1 d1)
                             '((quote PROG) (var data))
                             'DEFINT)
                    'DATA)
```

2nd Futamura Projection

```
(peval1 '(int p1 d1)                = ((peval2 '(peval1 '(int p1 d1)
        '(p1 d1)                                    '(p1 d1)
        '((quote PROG) (var data))                  (list (list 'quote p2) '(var data))
        'DEFINT)                                    'DEFINT)
                                            '(p2)
                                            '((var prog))
                                            'DEFPEVAL1)
                                    'PROG)
```

3rd Futamura Projection

```
(peval2 '(peval1 '(int p1 d1)       = ((peval3 '(peval2 '(peval1 '(int p1 d1)
                '(p1 d1)                                         '(p1 d1)
                (list (list 'quote p2)                           (list (list 'quote p2)
                      '(var data))                                     '(var data))
                'DEFINT)                                         i2)
        '(p2)                                            '(p2 i2)
        '((var prog))                                    (list '(var prog) (list 'quote i3))
        'DEFPEVAL1)                                      'DEFPEVAL1)
                                            '(i3)
                                            '((var defint))
                                            'DEFPEVAL2)
                                    'DEFINT)
```

Fig. 14: Futamura projections in Scheme.

Citation

ACM/SIGPLAN Workshop Partial Evaluation and Semantics-Based Program Manipulation >archive

Proceedings of the symposium on Partial evaluation and semantics-based program manipulation >toc
1991 , New Haven, Connecticut, United States

Towards multiple self-application

> Also published in ...

Author
Robert Glück

Sponsor
SIGPLAN : ACM Special Interest Group on Programming Languages

Publisher
ACM Press New York, NY, USA

Pages: 309 - 320 Series-Proceeding-Article
Year of Publication: 1991
ISSN:0362-1340

doi> http://doi.acm.org/10.1145/115865.115900 (Use this link to Bookmark this page)

Note: OCR errors may be found in this Reference List extracted from the full text article. ACM has opted to expose the complete List rather than only correct and linked references.

1 Bjcmer D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. North-Holland: Amsterdam 1988.

2 Bondorf A., Danvy O., Automatic autoprojection of recursive equations with global variables and abstract data types. Institute of Datalogy, University of Copenhagen. DIKU Report No. 90/4, 1990.

3 Bondorf A., Jones N. D., Mogensen T. AL et al., Binding time analysis and the taming of self-application. Institute of Datalogy, University of Copenhagen. Report 1988.

4 Consel C., New insights into partial evaluation: the Schism experiment. In: Ganzinger H. (ed.), ESOP '88. (Nancy, France). Lecture Notes in Computer Science, Vol. 300, 236- 246, Springer-Verlag 1988.

5 Charles Consel , Olivier Danvy, Partial evaluation of pattern matching in strings, Information Processing Letters, v.30 n.2, p.79-86, January 1989

6 Danvy O., Across the bridge between reflection and partial evaluation. In: Bj0rner D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. (Gammel Avermes, Denmark). 83-116, North-Holland 1988.

7 Ershov A. P., On the essence of compilation. In: Neuhold E. J. (ed.), Formal Description of Programming Concepts. 391- 420, North-Holland Publishing Co. 1978.

8 Futamura Y., Partial evaluation of computation process - an approach to a compiler-compiler. In: Systems, Computers, Controls, 2(5): 45-50, 1971.

9 Futamura Y., Nogi K., Generalized partial evaluation. In: BjCmer D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. (Gammel Averna~s, Denmark). 133- 151, Elsevier Science Publishers B. V. 1988.

10 Gliick R., Turchin V. F., Experiments with a self-applicable supercompiler. City University New York. Technical Report 1989,

11 R. Glueck , V. F. Turchin, Application of metasystem transition to function inversion and transformation, Proceedings of the international symposium on Symbolic and algebraic computation, p.286-287, August 20-24, 1990, Tokyo, Japan

12 Holst N. C. K., Language triplets: the Amix approach. In: Bj0mer D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. (Gammel Averna~s, Denmark). 167- 185, North-Holland 1988.

13 Neil D. Jones, Partial evaluation, self-application and types, Proceedings of the seventeenth international colloquium on Automata, languages and programming, p.639-659, July 1990, Warwick University, England

14 Jones N. D., Gomard C. K., Bondorf A. et al., A selfapplicable partial evaluator for the lambda calculus. In: IEEE Computer Society 1990 - International Conference on Computer Languages. (New Orleans, Louisiana). 1-10, IEEE Computer Society 1990.

15 Neil D. Jones , Peter Sestoft , Harald Sondergaard, An experiment in partial evaluation: the generation of a compiler generator, Proc. of the first international conference on Rewriting techniques and applications, p.124-140, December 1985, Dijon, France

16 Jones N. D., Sestoft P., SCndergaard H., Mix: a selfapplicable partial evaluator for experiments in compiler generation. In: Lisp and Symbolic Computation, 2(1): 9-50, 1989.

17 McCarthy J., Abrahams P. W., Edwards D. J. et al., Lisp 1.5 Programmer's Manual. MIT Press: Cambridge, Massachusetts 1962.

18 Mogensen T./E., Partially static structures in a self-applicable partial evaluator, in: Bjcrner D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. (Gammel Avemaes, Denmark). 325-347, North-Holland 1988.

19 Mogensen T. AL, Binding time aspects of partial evaluation. University of Copenhagen, Ph.D. Thesis, 1989.

20 J Rees , W Clinger, Revised report on the algorithmic language scheme, ACM SIGPLAN Notices, v.21 n.12, p.37-79, Dec. 1986

21 Romanenko S. A., A compiler generator produced by a selfapplicable specializer can have a surprisingly natural and understandable structure. In: Bjorner D., Ershov A. P., Jones N. D. (ed.), Partial Evaluation and Mixed Computation. (Gammel Averages, Denmark). 445-463, North-Holland 1988.

22 Peter. Sestoft, The structure of a self-applicable partial evaluator, on Programs as data objects, p.257-281, April 1986, Copenhagen, Denmark

23 Turchin V. F., The Phenomenon of Science. Columbia University Press: New York 1977.

24 Turchin V. F., A supercompiler system based on the language Refal. In: SIGPLAN Notices, 14(2): 46-54, 1979.

25 Turchin V. F., The language Refal - the theory of compilation and metasystem analysis. Courant Institute of Mathematical Sciences. Courant Computer Science Report No. 20, 1980.

26 Turchin V. F., The use of metasystem transition in theorem proving and program optimization. In: de Bakker J. W., van Leeuwen J. (ed.), Automata, Languages and Programming. (Noordwijkerhout, Netherlands), Lecture Notes in Computer Science, Vol. 85, 645-657, Springer-Verlag 1980.

27 Valentin F. Turchin, The concept of a supercompiler, ACM Transactions on Programming Languages and Systems (TOPLAS), v.8 n.3, p.292-325, July 1986

28 Turchin V. F., A constructive interpretation of the full set theory. In: The Journal of Symbolic Logic, 52(1): 172-201, 1987.

29 Turchin V. F., Refal-5, Programming Guide and Reference Manual. New England Publishing Co.: Holyoke, Massachusetts 1989.

↑ CITINGS *3*

Torben Æ. Mogensen, Self-applicable online partial evaluation of the pure lambda calculus, Proceedings of the ACM SIGPLAN Symposium on Partial evaluation and semantics-based program manipulation, p.39-44, June 21-23, 1995, La Jolla, California, United States

Rogardt Heldal , John Hughes, Partial evaluation and separate compilation, ACM SIGPLAN Notices, v.32 n.12, p.1-11, Dec. 1997

Renaud Marlet , Charles Consel , Philippe Boinot, Efficient incremental run-time specialization for free, ACM SIGPLAN Notices, v.34 n.5, p.281-292, May 1999

↑ **INDEX TERMS**

Primary Classification:
D. Software
↳ D.3 PROGRAMMING LANGUAGES

Additional Classification:
D. Software
↳ D.3 PROGRAMMING LANGUAGES
↳ D.3.2 Language Classifications

↳ Nouns: LISP

F. Theory of Computation
↳ F.3 LOGICS AND MEANINGS OF PROGRAMS

General Terms:
Languages, Theory

↑ **Peer to Peer - Readers of this Article have also read:**

Editorial pointers
Communications of the ACM 44, 9
Diane Crawford

News track
Communications of the ACM 44, 9
Robert Fox

Forum
Communications of the ACM 44, 9
Diane Crawford

New Products
Linux Journal 1996, 27es
CORPORATE Linux Journal Staff

Typechecking and modules for multi-methods
ACM SIGPLAN Notices 29, 10
Craig Chambers , Gary T. Leavens

↑ **This Article has also been published in:**

ACM SIGPLAN Notices
Volume 26 , Issue 9 (September 1991)

ACM, Inc.

Search Results

Search Results for: [thought system]
Found 7 of 110,178 searched. → Rerun within the Portal

Search within Results

matrix

GO > Advanced Search > Search Help/Tips

Sort by: **Title** **Publication** **Publication Date** **Score** Binder

Results 1 - 7 of 7 **short listing**

1 Viewpoint: existential education in the era of personal cybernetics 77%
Steve Mann
Communications of the ACM May 2000
Volume 43 Issue 5

2 Alternatives for modeling of preemptive scheduling 77%
James O. Henriksen
Proceedings of the 19th conference on Winter simulation December 1987
A system which includes overt instances of preemptive scheduling, or which requires the use of preemptive scheduling to model the system, can pose difficulties to a modeler. By preemptive scheduling we mean having to reschedule or cancel a previously scheduled event (or activity completion). The difficulties in modeling preemptive scheduling stem from the highly stylized constructs which simulation languages provide for such operations. This paper describes these difficulties, reviews an ap ...

3 User centered design in action: developing an intelligent agent application 77%
Jeanne Murray , David Schell , Cari Willis
Proceedings of the 15th annual international conference on Computer documentation October 1997

4 HCI and simulation packages 77%
Jasna Kuljis
Proceedings of the 28th conference on Winter simulation November 1996

5 Hypertext and the author/reader dialogue 77%
Susan Michalak , Mary Coney
Proceedings of the fifth ACM conference on Hypertext December 1993

6 Towards multiple self-application 77%
Robert Glück
ACM SIGPLAN Notices , Proceedings of the symposium on Partial evaluation and semantics-based program manipulation May 1991 Volume 26 Issue 9

7 Usability testing of a graphical programming system: things we missed in a programming walkthrough 77%
Brigham Bell , John Rieman , Clayton Lewis
Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology March 1991

Results 1 - 7 of 7 **short listing**

Search Results

Search Results for:
[displaying<AND>((transforming<AND>((generating<AND>((matrix<AND>((me
system))))))))]
Found 27 of 110,178 searched. → Rerun within the Portal

Search within Results

1 Incorporating usability into requirements engineering tools 77%
Gregory L. Smith , Sharon A. Stephens , Leonard L. Tripp , Wayne L. Warren
Proceedings of the ACM 1980 annual conference January 1980
The development of a computer system requires the definition of a precise set of properties or constraints that the system must satisfy with maximum economy and efficiency. This definition process requires a significant amount of communication between the requestor and the developer of the system. In recent years, several methodologies and tools have been proposed to improve this communication process. This paper establishes a framework for examining the methodologies and techniques, charti ...

2 A sensor simulation and animation system 77%
Timothy Zimmerlin , John Stanley , Warren Stone
Proceedings of the 5th annual conference on Computer graphics and interactive techniques August 1978
A shaded computer graphics system for sensor simulation and animation must possess extensive capabilities. Sensor simulation involves scenes that are viewed, environments that produce tones in the scenes, and sensors that image the scenes. Animation centers on image generation, but data input, motion specification,

and tone specification routines are very important for an effective system. A unique system has been developed that models a variety of scenes, environments, and senso ...

3 Typographic design for interfaces of information systems 77%
Aaron Marcus
Proceedings of the SIGCHI conference on Human factors in computing systems March 1982
Principles of information-oriented graphic design have been utilized in redesigning the interface for a large information management system. These principles are explained and examples of typical screen formats are shown to indicate the nature of improvements.

4 Query processing techniques in the summary-table-by-example database query language 77%
Gultekin Özsoyo?lu , Victor Matos , Meral Özsoyo?lu
ACM Transactions on Database Systems (TODS) December 1989
Volume 14 Issue 4
Summary-Table-by-Example (STBE) is a graphical language suitable for statistical database applications. STBE queries have a hierarchical subquery structure and manipulate summary tables and relations with set-valued attributes. The hierarchical arrangement of STBE queries naturally implies a tuple-by-tuple subquery evaluation strategy (similar to the nested loops join implementation technique) which may not be the best query processing strategy. In this paper we discuss the query ...

5 Concurrency control in groupware systems 77%
C. A. Ellis , S. J. Gibbs
ACM SIGMOD Record , Proceedings of the 1989 ACM SIGMOD international conference on Management of data June 1989
Volume 18 Issue 2
Groupware systems are computer-based systems that support two or more users engaged in a common task, and that provide an interface to a shared environment. These systems frequently require fine-granularity sharing of data and fast response times. This paper distinguishes real-time groupware systems from other multi-user systems and discusses their concurrency control requirements. An algorithm for concurrency control in real-time groupware systems is then presented. The advantages of this ...

6 A generalized model management system for mathematical programming 77%
Daniel R. Dolk

ACM Transactions on Mathematical Software (TOMS) September 1986
Volume 12 Issue 2

This paper examines mathematical programming software in the context of model management and decision support. The concept of a model management system (MMS) is introduced and compared to traditional modeling systems. An MMS is seen as a much more generalized software system that requires the confluence of existing operations research, database management, and artificial intelligence techniques. By incorporating powerful, abstraction-based representation structures, an MMS can support multi ...

7 High level knowledge sources in usable speech recognition systems 77%

S. L. Young , A. G. Hauptmann , W. H. Ward , E. T. Smith , P. Werner
Communications of the ACM February 1989
Volume 32 Issue 2

The authors detail an integrated system which combines natural language processing with speech understanding in the context of a problem solving dialogue. The MINDS system uses a variety of pragmatic knowledge sources to dynamically generate expectations of what a user is likely to say.

8 Session IX - coordination and decision making: Computer-based systems for cooperative work and group decisionmaking: status of use and problems in development 77%

Kenneth L. Kraemer , John Leslie King
Proceedings of the 1986 ACM conference on Computer-supported cooperative work December 1986

Application of computer and information technology to cooperative work and group decisionmaking has grown out of three traditions: computer-based communications, computer-based information service provision, and computer-based decision support. This paper provides an overview of the various kinds of systems that have been configured to meet the needs of groups at work, evaluates the status of these systems in the United States, evaluates the experience with them, assesses barriers to their furth ...

9 Applications experience with Linda 77%

Nicholas Carriero , David Gelernter
ACM SIGPLAN Notices , Proceedings of the ACM/SIGPLAN conference on Parallel programming: experience with applications, languages and systems January 1988
Volume 23 Issue 9

We describe three experiments using C-Linda to write parallel

codes. The first involves assessing the similarity of DNA sequences. The results demonstrate Linda's flexibility—Linda solutions are presented that work well at two quite different levels of granularity. The second uses a prime finder to illustrate a class of algorithms that do not (easily) submit to automatic parallelizers, but can be parallelized in straight-forward fashion using C-Linda. The final experiment describes th ...

10 Computer-based systems for cooperative work and group decision making 77%

Kenneth L. Kraemer , John Leslie King
ACM Computing Surveys (CSUR) July 1988
Volume 20 Issue 2

Application of computer and communications technology to cooperative work and group decision making has grown out of three traditions: computer-based communications, computer:based information service provision, and computer-based decision support. This paper reviews the group decision support systems (GDSSs) that have been configured to meet the needs of groups at work, and evaluates the experience to date with such systems. Progress with GDSSs has proved to be slower than originally antic ...

11 Homomorphic factorization of BRDFs for high-performance rendering 77%

Michael D. McCool , Jason Ang , Anis Ahmad
Proceedings of the 28th annual conference on Computer graphics and interactive techniques August 2001

A bidirectional reflectance distribution function (BRDF) describes how a material reflects light from its surface. To use arbitrary BRDFs in real-time rendering, a compression technique must be used to represent BRDFs using the available texture-mapping and computational capabilities of an accelerated graphics pipeline. We present a numerical technique, homomorphic factorization, that can decompose arbitrary BRDFs into products of two or more factors of lower dimensionality, each factor de ...

12 Integration and applications of the TAU performance system in parallel Java environments 77%

Sameer Shende , Allen D. Malony
Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande
June 2001

Parallel Java environments present challenging problems for performance tools because of Java's rich language system and its multi-level execution platform combined with the integration of native-code application libraries and parallel runtime software. In addition to the desire to provide robust performance measurement and analysis capabilities for the Java language itself, the coupling of different software execution contexts under a uniform performance model needs careful consideration of ...

13 Curriculum 68: Recommendations for academic programs in computer science: a report of the ACM curriculum committee on computer science 77%
William F. Atchison , Samuel D. Conte , John W. Hamblen , Thomas E. Hull , Thomas A. Keenan , William B. Kehl , Edward J. McCluskey , Silvio O. Navarro , Werner C. Rheinboldt , Earl J. Schweppe , William Viavant , David M. Young
Communications of the ACM March 1968
Volume 11 Issue 3

14 Answering English questions by computer: a survey 77%
R. F. Simmons
Communications of the ACM January 1965
Volume 8 Issue 1

15 ACM forum 77%
Robert L. Ashenhurst
Communications of the ACM February 1975
Volume 18 Issue 2

16 Computer-aided analysis and design of information systems 77%
J. F. Nunamaker , Benn R. Konsynski , Thomas Ho , Carl Singer
Communications of the ACM December 1976
Volume 19 Issue 12
This paper describes the use of computer-aided analysis for the design and development of an integrated financial management

system by the Navy Material Command Support Activity (NMCSA). Computer-aided analysis consists of a set of procedures and computer programs specifically designed to aid in the process of applications software design, computer selection and performance evaluation. There are four major components: Problem Statement Language, Problem Statement Analyzer, Generator of Alte ...

17 Curriculum '78: recommendations for the undergraduate program in computer science— a report of the ACM curriculum committee on computer science 77%

Richard H. Austing , Bruce H. Barnes , Della T. Bonnette , Gerald L. Engel , Gordon Stokes
Communications of the ACM March 1979
Volume 22 Issue 3

Contained in this report are the recommendations for the undergraduate degree program in Computer Science of the Curriculum Committee on Computer Science (C3S) of the Association for Computing Machinery (ACM). The core curriculum common to all computer science undergraduate programs is presented in terms of elementary level topics and courses, and intermediate level courses. Elective courses, used to round out an undergraduate program, are then discussed, and ...

18 Program Transformation Systems 77%

H. Partsch , R. Steinbrüggen
ACM Computing Surveys (CSUR) September 1983
Volume 15 Issue 3

19 The Logical Record Access Approach to Database Design 77%

Toby J. Teorey , James P. Fry
ACM Computing Surveys (CSUR) June 1980
Volume 12 Issue 2

20 Database Management Systems Development in the USSR 77%

A. G. Dale
ACM Computing Surveys (CSUR) September 1979
Volume 11 Issue 3

Results 1 - 20 of 27 **short listing**

Prev Page 1 2 Next Page

Inc.

Search Results

Search Results for:
[displaying<AND>((transforming<AND>((generating<AND>((matrix<AND>((me
system))))))))]
Found 27 of 110,178 searched. → Rerun within the Portal

Search within Results

21 Geographic Data Processing 77%
George Nagy , Sharad Wagle
ACM Computing Surveys (CSUR) June 1979
Volume 11 Issue 2

22 The Computer in the Humanities and Fine Arts 77%
Sally Yeates Sedelow
ACM Computing Surveys (CSUR) June 1970
Volume 2 Issue 2

23 Visualizing modeling heuristics: an exploratory study 77%
Laurie B. Waisel , William A. Wallace , Thomas R. Willemain
Proceeding of the 20th international conference on Information Systems January 1999

24 Symbolic mathematical computation 77%
Stephen Wolfram
Communications of the ACM April 1985
Volume 28 Issue 4
Standard programming languages are inadequate for the kind of symbolic mathematical computations that theoretical physicists

need to perform. Higher mathematics systems like SMP address this problem.

25 Databsse system approach the management decision support 77%
John J. Donovan
ACM Transactions on Database Systems (TODS) December 1976
Volume 1 Issue 4
Traditional intuitive methods of decision-making are no longer adequate to deal with the complex problems faced by the modern policymaker. Thus systems must be developed to provide the information and analysis necessary for the decisions which must be made. These systems are called decision support systems. Although database systems provide a key ingredient to decision support systems, the problems now facing the policymaker are different from those problems to which database systems have b ...

26 An analysis of geometric modeling in database systems 77%
Alfons Kemper , Mechtild Wallrath
ACM Computing Surveys (CSUR) March 1987
Volume 19 Issue 1
The data-modeling and computational requirements for integrated computer aided manufacturing (CAM) databases are analyzed, and the most common representation schemes for modeling solid geometric objects in a computer are described. The *primitive instancing* model, the *boundary representation,* and the *constructive solid geometry* model are presented from the viewpoint of database representation. Depending on the representation scheme, one can apply geometric transformation ...

27 Launching the new era 77%
Kazuhiro Fuchi , Robert Kowalski , Koichi Furukawa , Kazunori Ueda , Ken Kahn , Takashi Chikayama , Evan Tick
Communications of the ACM March 1993
Volume 36 Issue 3

Results 21 - 27 of 27 **short listing**

Prev Page 1 2 Next Page

Search Results

Search Results for:
[displaying<AND>((transforming<AND>((generating<AND>((matrix<AND>((me
system))))))))]
Found 27 of 110,178 searched. → Rerun within the Portal

Search within Results

1 Incorporating usability into requirements engineering tools 77%
Gregory L. Smith , Sharon A. Stephens , Leonard L. Tripp , Wayne L. Warren
Proceedings of the ACM 1980 annual conference January 1980
The development of a computer system requires the definition of a precise set of properties or constraints that the system must satisfy with maximum economy and efficiency. This definition process requires a significant amount of communication between the requestor and the developer of the system. In recent years, several methodologies and tools have been proposed to improve this communication process. This paper establishes a framework for examining the methodologies and techniques, charti ...

2 A sensor simulation and animation system 77%
Timothy Zimmerlin , John Stanley , Warren Stone
Proceedings of the 5th annual conference on Computer graphics and interactive techniques August 1978
A shaded computer graphics system for sensor simulation and animation must possess extensive capabilities. Sensor simulation involves scenes that are viewed, environments that produce tones in the scenes, and sensors that image the scenes. Animation centers on image generation, but data input, motion specification,

and tone specification routines are very important for an effective system. A unique system has been developed that models a variety of scenes, environments, and senso ...

3 Typographic design for interfaces of information systems 77%
Aaron Marcus
Proceedings of the SIGCHI conference on Human factors in computing systems March 1982
Principles of information-oriented graphic design have been utilized in redesigning the interface for a large information management system. These principles are explained and examples of typical screen formats are shown to indicate the nature of improvements.

4 Query processing techniques in the summary-table-by-example database query language 77%
Gultekin Özsoyo?lu , Victor Matos , Meral Özsoyo?lu
ACM Transactions on Database Systems (TODS) December 1989
Volume 14 Issue 4
Summary-Table-by-Example (STBE) is a graphical language suitable for statistical database applications. STBE queries have a hierarchical subquery structure and manipulate summary tables and relations with set-valued attributes. The hierarchical arrangement of STBE queries naturally implies a tuple-by-tuple subquery evaluation strategy (similar to the nested loops join implementation technique) which may not be the best query processing strategy. In this paper we discuss the query ...

5 Concurrency control in groupware systems 77%
C. A. Ellis , S. J. Gibbs
ACM SIGMOD Record , Proceedings of the 1989 ACM SIGMOD international conference on Management of data June 1989
Volume 18 Issue 2
Groupware systems are computer-based systems that support two or more users engaged in a common task, and that provide an interface to a shared environment. These systems frequently require fine-granularity sharing of data and fast response times. This paper distinguishes real-time groupware systems from other multi-user systems and discusses their concurrency control requirements. An algorithm for concurrency control in real-time groupware systems is then presented. The advantages of this ...

6 A generalized model management system for mathematical programming 77%
Daniel R. Dolk

ACM Transactions on Mathematical Software (TOMS) September 1986
Volume 12 Issue 2

This paper examines mathematical programming software in the context of model management and decision support. The concept of a model management system (MMS) is introduced and compared to traditional modeling systems. An MMS is seen as a much more generalized software system that requires the confluence of existing operations research, database management, and artificial intelligence techniques. By incorporating powerful, abstraction-based representation structures, an MMS can support multi ...

7 High level knowledge sources in usable speech recognition systems 77%

S. L. Young , A. G. Hauptmann , W. H. Ward , E. T. Smith , P. Werner
Communications of the ACM February 1989
Volume 32 Issue 2

The authors detail an integrated system which combines natural language processing with speech understanding in the context of a problem solving dialogue. The MINDS system uses a variety of pragmatic knowledge sources to dynamically generate expectations of what a user is likely to say.

8 Session IX - coordination and decision making: Computer-based systems for cooperative work and group decisionmaking: status of use and problems in development 77%

Kenneth L. Kraemer , John Leslie King
Proceedings of the 1986 ACM conference on Computer-supported cooperative work December 1986

Application of computer and information technology to cooperative work and group decisionmaking has grown out of three traditions: computer-based communications, computer-based information service provision, and computer-based decision support. This paper provides an overview of the various kinds of systems that have been configured to meet the needs of groups at work, evaluates the status of these systems in the United States, evaluates the experience with them, assesses barriers to their furth ...

9 Applications experience with Linda 77%

Nicholas Carriero , David Gelernter
ACM SIGPLAN Notices , Proceedings of the ACM/SIGPLAN conference on Parallel programming: experience with applications, languages and systems January 1988
Volume 23 Issue 9

We describe three experiments using C-Linda to write parallel

codes. The first involves assessing the similarity of DNA sequences. The results demonstrate Linda's flexibility—Linda solutions are presented that work well at two quite different levels of granularity. The second uses a prime finder to illustrate a class of algorithms that do not (easily) submit to automatic parallelizers, but can be parallelized in straight-forward fashion using C-Linda. The final experiment describes th ...

10 Computer-based systems for cooperative work and group decision making 77%

Kenneth L. Kraemer , John Leslie King
ACM Computing Surveys (CSUR) July 1988
Volume 20 Issue 2

Application of computer and communications technology to cooperative work and group decision making has grown out of three traditions: computer-based communications, computer:based information service provision, and computer-based decision support. This paper reviews the group decision support systems (GDSSs) that have been configured to meet the needs of groups at work, and evaluates the experience to date with such systems. Progress with GDSSs has proved to be slower than originally antic ...

11 Homomorphic factorization of BRDFs for high-performance rendering 77%

Michael D. McCool , Jason Ang , Anis Ahmad
Proceedings of the 28th annual conference on Computer graphics and interactive techniques August 2001

A bidirectional reflectance distribution function (BRDF) describes how a material reflects light from its surface. To use arbitrary BRDFs in real-time rendering, a compression technique must be used to represent BRDFs using the available texture-mapping and computational capabilities of an accelerated graphics pipeline. We present a numerical technique, homomorphic factorization, that can decompose arbitrary BRDFs into products of two or more factors of lower dimensionality, each factor de ...

12 Integration and applications of the TAU performance system in parallel Java environments 77%

Sameer Shende , Allen D. Malony
Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande
June 2001

Parallel Java environments present challenging problems for performance tools because of Java's rich language system and its multi-level execution platform combined with the integration of native-code application libraries and parallel runtime software. In addition to the desire to provide robust performance measurement and analysis capabilities for the Java language itself, the coupling of different software execution contexts under a uniform performance model needs careful consideration of ...

13 Curriculum 68: Recommendations for academic programs in computer science: a report of the ACM curriculum committee on computer science 77%
William F. Atchison , Samuel D. Conte , John W. Hamblen , Thomas E. Hull , Thomas A. Keenan , William B. Kehl , Edward J. McCluskey , Silvio O. Navarro , Werner C. Rheinboldt , Earl J. Schweppe , William Viavant , David M. Young
Communications of the ACM March 1968
Volume 11 Issue 3

14 Answering English questions by computer: a survey 77%
R. F. Simmons
Communications of the ACM January 1965
Volume 8 Issue 1

15 ACM forum 77%
Robert L. Ashenhurst
Communications of the ACM February 1975
Volume 18 Issue 2

16 Computer-aided analysis and design of information systems 77%
J. F. Nunamaker , Benn R. Konsynski , Thomas Ho , Carl Singer
Communications of the ACM December 1976
Volume 19 Issue 12
This paper describes the use of computer-aided analysis for the design and development of an integrated financial management

system by the Navy Material Command Support Activity (NMCSA). Computer-aided analysis consists of a set of procedures and computer programs specifically designed to aid in the process of applications software design, computer selection and performance evaluation. There are four major components: Problem Statement Language, Problem Statement Analyzer, Generator of Alte ...

17 Curriculum '78: recommendations for the undergraduate program in computer science— a report of the ACM curriculum committee on computer science 77%
Richard H. Austing , Bruce H. Barnes , Della T. Bonnette , Gerald L. Engel , Gordon Stokes
Communications of the ACM March 1979
Volume 22 Issue 3
Contained in this report are the recommendations for the undergraduate degree program in Computer Science of the Curriculum Committee on Computer Science (C3S) of the Association for Computing Machinery (ACM). The core curriculum common to all computer science undergraduate programs is presented in terms of elementary level topics and courses, and intermediate level courses. Elective courses, used to round out an undergraduate program, are then discussed, and ...

18 Program Transformation Systems 77%
H. Partsch , R. Steinbrüggen
ACM Computing Surveys (CSUR) September 1983
Volume 15 Issue 3

19 The Logical Record Access Approach to Database Design 77%
Toby J. Teorey , James P. Fry
ACM Computing Surveys (CSUR) June 1980
Volume 12 Issue 2

20 Database Management Systems Development in the USSR 77%
A. G. Dale
ACM Computing Surveys (CSUR) September 1979
Volume 11 Issue 3

Results 1 - 20 of 27 **short listing**

Prev Page 1 2 Next Page

Inc.

WEST

Generate Collection | Print

Search Results - Record(s) 1 through 7 of 7 returned.

☐ 1. Document ID: US 20020089551 A1

L9: Entry 1 of 7 File: PGPB Jul 11, 2002

PGPUB-DOCUMENT-NUMBER: 20020089551
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020089551 A1

TITLE: Method and apparatus for displaying a thought network from a thought's perspective

PUBLICATION-DATE: July 11, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Hugh, Harlan M.	Los Angeles	CA	US	
Crawford, Jenson	Los Angeles	CA	US	

US-CL-CURRENT: 345/853

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

☐ 2. Document ID: US 20020067381 A1

L9: Entry 2 of 7 File: PGPB Jun 6, 2002

PGPUB-DOCUMENT-NUMBER: 20020067381
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020067381 A1

TITLE: Method and apparatus for organizing and processing information using a digital computer

PUBLICATION-DATE: June 6, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Hugh, Harlan M.	Los Angeles	CA	US	

US-CL-CURRENT: 345/854; 345/748

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

☐ 3. Document ID: US 20020054167 A1

L9: Entry 3 of 7 File: PGPB May 9, 2002

PGPUB-DOCUMENT-NUMBER: 20020054167
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020054167 A1

TITLE: Method and apparatus for filtering and displaying a thought network from a thought's perspective

PUBLICATION-DATE: May 9, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Hugh, Harlan M.	Los Angeles	CA	US	

US-CL-CURRENT: 345/854

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

---

☐ 4. Document ID: US 6256032 B1

L9: Entry 4 of 7 File: USPT Jul 3, 2001

US-PAT-NO: 6256032
DOCUMENT-IDENTIFIER: US 6256032 B1

TITLE: Method and apparatus for organizing and processing information using a digital computer

DATE-ISSUED: July 3, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hugh; Harlan M.	Los Angeles	CA		

US-CL-CURRENT: 345/854; 345/764

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 5. Document ID: US 6166739 A

L9: Entry 5 of 7 File: USPT Dec 26, 2000

US-PAT-NO: 6166739
DOCUMENT-IDENTIFIER: US 6166739 A

TITLE: Method and apparatus for organizing and processing information using a digital computer

DATE-ISSUED: December 26, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hugh; Harlan M.	Los Angeles	CA		

US-CL-CURRENT: 345/854; 345/858, 709/100

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments KWIC | Draw Desc | Image

☐ 6. Document ID: US 6031537 A

L9: Entry 6 of 7 File: USPT Feb 29, 2000

US-PAT-NO: 6031537
DOCUMENT-IDENTIFIER: US 6031537 A

TITLE: Method and apparatus for displaying a thought network from a thought's perspective

DATE-ISSUED: February 29, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hugh; Harlan M.	Los Angeles	CA		

US-CL-CURRENT: 345/854; 345/839

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments KWIC | Draw Desc | Image

☐ 7. Document ID: EP 1103901 A2

L9: Entry 7 of 7 File: EPAB May 30, 2001

PUB-NO: EP001103901A2
DOCUMENT-IDENTIFIER: EP 1103901 A2
TITLE: Method and apparatus for analyzing thought system

PUBN-DATE: May 30, 2001

INVENTOR-INFORMATION:

NAME	COUNTRY
SUZUKI, KAZUHIKO	JP

INT-CL (IPC): G06 F 17/30

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments KWIC | Draw Desc | Image

Generate Collection | Print

Terms	Documents
L6 and generating and transforming and displaying	7

Display Format: - Change Format

Previous Page Next Page